FACILITANDO LA ADQUISICIÓN DE HABILIDADES BÁSICAS EN CURSOS DE PROCESADORES DE LENGUAJE: UN ENFOQUE APOYADO EN VIDEOJUEGOS

DANIEL RODRIGUEZ CEREZO

MÁSTER EN INVESTIGACIÓN EN INFORMÁTICA, FACULTAD DE INFORMÁTICA,
UNIVERSIDAD COMPLUTENSE DE MADRID

Trabajo Fin Máster en Sistemas Inteligentes

Junio de 2011
Calificación: Sobresaliente

Director:

José Luis Sierra Rodríguez

Autorización de Difusión

DANIEL RODRÍGUEZ CEREZO

20 de Junio de 2011

El abajo firmante, matriculado en el Máster en Investigación en Informática de la Facultad de Informática, autoriza a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor el presente Trabajo Fin de Máster: "FACILITANDO LA ADQUISICIÓN DE HABILIDADES BÁSICAS EN CURSOS DE PROCESADORES DE LENGUAJE: UN ENFOQUE APOYADO EN VIDEOJUEGOS", realizado durante el curso académico 2010-2011 bajo la dirección de José Luis Sierra Rodríguez en el Departamento de Ingeniería del Software e Inteligencia Artificial, y a la Biblioteca de la UCM a depositarlo en el Archivo Institucional E-Prints Complutense con el objeto de incrementar la difusión, uso e impacto del trabajo en Internet y garantizar su preservación y acceso a largo plazo.

Resumen en castellano

Evaluators es un sistema educativo que permite a los docentes que imparten las materias relacionadas con Procesadores de Lenguajes generar videojuegos a partir de colecciones de ejercicios relacionados con tareas de procesamiento de lenguaje. Para este propósito dicho sistema adopta el formalismo de las *gramáticas de atributos* como modelo central de una traducción dirigida por la sintaxis y usa una metáfora adecuada para mapear los árboles sintácticos decorados y la evaluación semántica en videojuegos. Estos videojuegos proporcionan al estudiante una experiencia inmersiva que los ayuda a comprender los conceptos fundamentales subyacentes a las gramáticas de atributos de manera sencilla y amena. *Evaluators* ha sido desarrollado siguiendo un modelo de proceso para la construcción de herramientas educativas dirigido por los ejercicios propuestos por los docentes. Este modelo supone una estrecha colaboración entre los docentes y los desarrolladores. Además, permite incorporar de manera coherente diferentes evaluaciones formativas orientadas a mejorar la calidad de los sistemas desarrollados.

Palabras clave

Educación en Procesadores de Lenguajes, Herramientas Educativas, Aprendizaje basado en Videojuegos, Gramáticas de Atributos, Modelo de Proceso de Desarrollo, Desarrollo de Software Dirigido por Lenguajes

Resumen en inglés

Evaluators is an educational system that lets instructors in computer language implementation courses generate videogames from collections of exercises concerning language processing tasks. For this purpose, the system adopts *attribute grammars* as a central model of syntax-directed translation and uses an appropriate metaphor to map attributed syntax trees and semantic evaluation into videogames. These games provide students with an immersive experience which helps them to better comprehend the fundamental concepts behind attribute grammars in an entertaining way. The development of *Evaluators* follows a process model for the construction of educational games, which is driven by the exercises proposed by the instructors. This model promotes a tight collaboration between instructors and game developers during game construction. Also it makes it possible to include several formative evaluations oriented to improve the quality of the developed system.

Keywords

Education in Language Processing, Educational Tools, Game-based learning, Attribute Grammars, Development Process Model, Language-Driven Software Development.

Índice de contenidos

Agradecimientos

En primer lugar quiero agradecer a José Luis el esfuerzo dedicado para conseguir sacar adelante este trabajo de fin de máster. Sin sus horas de dedicación no podría haber llegado tan lejos. Quiero agradecer también a Mercedes sus consejos para mejorar el sistema y su colaboración en las tareas de evaluación. También agradecer a los alumnos de la clase de Procesadores de Lenguaje del grupo A del curso 2010-2011 que participaron en el estudio que realizamos. Agradecer también a Rafael y Ángel su colaboración en los primeros pasos de desarrollo del sistema presentado en este trabajo de máster. Sin su esfuerzo durante el proyecto de fin de carrera este proyecto no podría haber sido posible. Y por último agradecer a mi familia su apoyo y comprensión durante este año.

Capítulo 1. Introducción

Las principales recomendaciones de los currículos para enseñanzas universitarias en Informática reconocen las materias impartidas en *Procesadores de Lenguajes* como un elemento clave de dichas enseñanzas [1]. De esta forma, estas materias aparecen tradicionalmente incluidas en los programas educativos de las titulaciones en Informática de la mayor parte de las universidades.

En la Facultad de Informática de la Universidad Complutense de Madrid, los docentes son conscientes de la importancia de separar claramente los aspectos relativos a la especificación y la implementación de los procesadores a la hora de impartir las materias que componen Procesadores de Lenguajes. Por ello, los docentes que imparten estas materias optan por emplear la *traducción dirigida por la sintaxis* [3] como paradigma de construcción de procesadores y el formalismo de las *gramáticas de atributos* [38][39][49] como modelo básico para la especificación de las tareas a realizar en dicho paradigma.

La estrategia seguida para enseñar el funcionamiento de las gramáticas de atributos por parte de los docentes de la UCM se basa en un sistema donde el docente apoya sus clases teóricas con el planteamiento y resolución de baterías de ejercicios relacionados con dicho formalismo. Aunque este sistema basado en ejercicios se ha mostrado útil, los docentes han observado también que los alumnos no llegan a entender en profundidad el formalismo, sino que estos se limitan a aprender de forma algorítmica y rutinaria el funcionamiento de dicho formalismo. La raíz de este problema se encuentra, parcialmente, en que los alumnos no son capaces de asimilar los conceptos básicos subyacentes a dicho formalismo en etapas tempranas de la formación, ni tampoco los relativos al modelo de proceso derivado de dicho formalismo, modelo de proceso que está dirigido por las dependencias entre los atributos, en lugar de por un proceso lineal, como el de análisis sintáctico.

Por otro lado, los videojuegos educativos han probado ser poderosas herramientas para la enseñanza y se centran principalmente en la enseñanza de los procesos mentales más que en los conceptos [25]. Para este propósito, este tipo de videojuegos suelen emplazar al estudiante en un mundo virtual diseñado para guiarle mientras resuelve el problema y ofrecerle experiencias

educativas de calidad. Además, han demostrado ser capaces de motivar a los estudiantes a utilizarlos y mejorar su aprendizaje mientras se divierten.

Por ello, en este trabajo de fin de máster se propone aunar ambos enfoques: el dirigido por ejercicios propugnado por los docentes de Procesadores de Lenguaje, y el enfoque didáctico basado en videojuegos. Para ello se propone diseñar un sistema viable y aplicable en la práctica para la producción de videojuegos educativos destinados a enseñar el modelo de proceso subyacente al formalismo de las gramáticas de atributos de una forma entretenida y pedagógicamente sólida, a partir de baterías de ejercicios cuidadosamente seleccionados.

1.1 Objetivos del Proyecto

El objetivo primario de este proyecto es llevar a cabo una investigación sobre mecanismos basados en juegos para la enseñanza de conceptos básicos en procesadores de lenguaje, y, en particular, de conceptos relativos al formalismo de las gramáticas de atributos. Este objetivo primario se desgrana, a su vez, en los siguientes objetivos específicos:

- Realizar un estudio del estado del arte en: (i) métodos de enseñanza de la materia de procesadores de lenguaje, (ii) herramientas educativas en informática y, en particular, herramientas educativas para la enseñanza de procesadores de lenguaje, y (iii) herramientas basadas en juegos para la enseñanza y el aprendizaje de la informática.
- Proponer un modelo de proceso genérico para la construcción de sistemas educativos basados en juegos, y dirigidos por ejercicios, para la construcción de herramientas de enseñanza / aprendizaje en el dominio de la Informática.
- Aplicar la experiencia ganada con el estudio del estado de la cuestión, así como del modelo de proceso genérico formulado, a la construcción del sistema educativo para la enseñanza de conceptos básicos sobre gramáticas de atributos. Dicho sistema se construirá, así mismo, mediante la aplicación de una serie de mejoras soportadas por un proceso de evaluación formativa con usuarios (profesores y alumnos) a la versión preliminar del sistema para la producción de juegos educativos orientados a la enseñanza de gramáticas de atributos descrito en [22] (dicho sistema fue desarrollado parcialmente por el autor como parte de su

proyecto de Sistemas Informáticos en la titulación de Ingeniería Informática de la UCM).

Como resultado práctico de estos objetivos, se obtendrá una versión substancialmente mejorada del prototipo creado en [22] para acercarlo más a un producto software final, que resulte útil tanto para docentes como para estudiantes.

1.2 Estructura del Documento

La memoria de este Proyecto de Investigación está estructurada de la siguiente manera:

- En el capítulo 2 se lleva a cabo una revisión sobre el estado de la cuestión de los distintos temas tratados en este proyecto, introduciendo los métodos de enseñanza de las materias de Procesadores de Lenguaje, así como distintas herramientas software educativas en el campo de la Ingeniería Informática, y más concretamente, en las materias de Procesadores de Lenguaje. También se revisan distintos videojuegos educativos aplicados al campo de la Ingeniería Informática. Como resultado se satisface el primer objetivo específico presentado anteriormente. En este capítulo se introduce también el formalismo de las gramáticas de atributos, a fin de conseguir que esta memoria sea lo más autocontenida posible.
- En el capítulo 3 se expone el sistema educativo propuesto: *Evaluators*. En primer lugar, se describe el modelo de proceso genérico seguido para la creación del sistema. Después se muestra cómo se ha aplicado dicho modelo para desarrollar dicho sistema educativo. Como consecuencia, este capítulo satisface el segundo y, parcialmente, el tercero de los objetivos planteados.
- Por último, el capítulo 4 describe distintas experiencias de evaluación realizadas con el sistema propuesto y la versión previa del mismo. Dichas experiencias involucran tanto a docentes como a estudiantes. El capítulo describe también las consecuencias de las mismas en la evolución del propio sistema, completando, por tanto, la consecución del tercer objetivo propuesto.
- Para concluir, en el capítulo 5 se explican las conclusiones finales, así como las posibles líneas de trabajo futuro a seguir para mejorar el sistema. La memoria se

cierra con distintos apéndices que complementan los contenidos desarrollados en la misma.

Capítulo 2. Estado de la Cuestión

En este capítulo se realiza una descripción del campo de investigación de este proyecto de fin de máster. En primer lugar, ya que el proyecto presentado se centra en mejorar la experiencia educativa de los alumnos en el desarrollo de materias relacionadas con el procesamiento de lenguajes informáticos, se revisan distintos métodos que se siguen a la hora de enseñar los conceptos que componen esta materia. Posteriormente se revisan una serie de herramientas software destinadas a la enseñanza de distintos aspectos que se imparten a lo largo de la carrera de Ingeniería Informática. Después se analizan una serie de herramientas educativas destinadas a la materia de procesamiento de lenguaje, con el fin de fijar el contexto necesario para situar *Evaluators* como herramienta educativa dentro de la misma. Por otro lado, ya que *Evaluators* es un sistema basado en videojuegos para facilitar la enseñanza y el aprendizaje de mecanismos básicos en procesamiento de lenguaje, se investiga la utilidad de los videojuegos como herramienta de educación y se hace una revisión de distintos videojuegos educativos. Por último, dado que el modelo adoptado en *Evaluators* es el de las *gramáticas de atributos*, se finaliza este capítulo explicando brevemente los principios básicos de este formalismo.

2.1 Enseñanza de la Materia de Procesadores de Lenguaje

Las materias relacionadas con el diseño y la implementación de lenguajes informáticos han sido tradicionalmente materias complicadas para los estudiantes, principalmente debido a dos motivos:

- A pesar de la utilidad que tiene la materia como soporte a los nuevos enfoques de desarrollo de software dirigidos por modelos y por lenguajes específicos de dominio [23][37], los alumnos no son conscientes de dicha utilidad para su vida profesional, y por lo tanto no se encuentran motivados a la hora de abordar el estudio de la materia [72].
- Por otra parte, en estas materias los alumnos deben realizar proyectos de gran tamaño, como por ejemplo implementar un traductor para un lenguaje de programación no trivial, lo que suele provocar que muchos alumnos abandonen la asignatura o la suspendan por no ser capaces de alcanzar este objetivo con éxito [32].

Por otro lado, y tal y como apunta Alfred V. Aho en [2], esta materia siempre ha sido difícil de enseñar, debido al enorme arsenal de técnicas y métodos abordados en la misma. Por tanto, tradicionalmente se han aplicado distintas estrategias pedagógicas orientadas a facilitar y sistematizar dicha enseñanza. A continuación se describen tres de estas estrategias:

- Un enfoque basado en proyectos, que propugna la creación de un compilador para un lenguaje de programación reducido.
- Un enfoque basado en la creación de pequeños lenguajes de programación que ilustren los diferentes aspectos del diseño e implementación de lenguajes informáticos.
- Un enfoque basado en el análisis y depuración de un compilador real para un lenguaje de programación completo.

2.1.1 Implementación de un procesador para un lenguaje de programación reducido

Esta estrategia es la más habitualmente utilizada por una gran parte de los docentes. Los docentes proponen un lenguaje de programación reducido y los alumnos se encargan de implementar un compilador capaz de traducir a un lenguaje objeto adecuado los programas especificados por el lenguaje. En la literatura es posible encontrar distintos lenguajes reducidos prototípicos, utilizados como base a esta estrategia (por ejemplo, COOL [4], MINIML [9] o CHIRP [78]). Estos lenguajes se caracterizan por tener un par de tipos básicos, un repertorio básico de operaciones entre estos tipos, bucles, condicionales y otras estructuras de control, tipos definidos por el programador (vectores, registros, etc.), y algún mecanismo de abstracción (normalmente procedimientos y/o funciones).

Aparte de basar la estrategia en lenguajes de programación, otros docentes abogan por utilizar lenguajes menos convencionales como, por ejemplo:

- Lenguajes de representación de grafos [73], que permiten describir grafos que, posteriormente, pueden ser dibujados por el procesador de lenguaje construido por el alumno.
- Lenguajes de dibujado de figuras [57], donde el procesador de lenguaje se encarga de transformar las especificaciones de gráficos en dibujos formados a partir de figuras simples (cuadrados, triángulos, círculos, etc.).

- Lenguajes de programación de robots [78] donde los alumnos crean un procesador de lenguaje que analizará un lenguaje especificado por el docente para enviar órdenes a un robot.

La ventaja de utilizar este tipo de lenguajes específicos, frente a lenguajes de programación más convencionales, es el resultar más atractivos para los alumnos, lo que puede motivarles a implicarse más en el proyecto y aprender mejor los conceptos sobre procesamiento de lenguaje presentados en clase.

En la aplicación de la estrategia basada en proyectos también se encuentran diferencias a la hora de implementar el lenguaje propuesto. De esta forma, algunos docentes apuestan por obligar a sus alumnos a implementar cada una de las partes del compilador desde cero, como por ejemplo los trabajos expuestos en [21][59], mientras que otros proporcionan a los mismos un conjunto básico de utilidades (por ejemplo, componentes para manejar la tabla de símbolos, para generar código, etc.) [8]. Por otro lado, otros docentes intentan enseñar a los alumnos la importancia de apoyarse en herramientas a la hora de programar los procesadores, permitiendo a los mismos el escribir sus compiladores utilizando herramientas de generación automática, tales como YACC, BISON o JavaCC (véase, por ejemplo, [17][43]). Esto permite a los alumnos centrarse en las nociones teóricas de la materia, y no perder el tiempo con problemas de implementación de más bajo nivel.

2.1.2 Pequeños proyectos de procesamiento de lenguaje

Esta estrategia nace para resolver los problemas que origina en los alumnos la ejecución de un proyecto de gran envergadura, como es el desarrollo de un compilador para un lenguaje no trivial:

- Algunos alumnos se traban con las primeras fases de la construcción del compilador y no son capaces de completar el proyecto con éxito en el tiempo exigido.
- Además, algunos de los conceptos necesarios para implementar el compilador no son impartidos hasta bien avanzado el curso, lo que limita aún más el tiempo del que disponen los alumnos para implementar dicho programa.

Por ello, la estrategia basada en pequeños proyectos de procesamiento de lenguaje aboga por proponer a los alumnos pequeños proyectos relativos a distintos lenguajes que estén

íntimamente ligados con los conceptos vistos en clase. Así, a medida que el curso avanza, son capaces de aplicar inmediatamente los conceptos vistos en clase a dichos microlenguajes. Por otro lado, y con el fin de centrar cada proyecto en la materia de interés en ese punto del curso, los docentes suelen ofrecerles partes ya implementadas de los compiladores correspondientes (por ejemplo, el analizador léxico y/o el sintáctico). En los trabajos presentados en [41][62] se analiza esta estrategia con mayor profundidad.

2.1.3 Análisis y depuración de un procesador de lenguaje real

La tesis principal de esta estrategia es que tanto la implementación de un compilador para un lenguaje reducido, como la implementación de pequeños compiladores no es suficiente para enseñar a los estudiantes todos los conceptos involucrados en un procesador de lenguaje real, para un lenguaje informático real.

Basándose en dicha tesis, en [75] se propone enseñar el funcionamiento de los procesadores de lenguaje y los compiladores mediante la depuración del código fuente de un compilador profesional. Más concretamente, en el trabajo citado, se opta por usar un compilador para C#. Los alumnos son dirigidos en sesiones de laboratorio para, mediante el uso de *breakpoints* y la visualización de determinadas variables, observar los pasos llevados a cabo en determinados momentos del análisis y traducción de un programa en C#, lo que permite complementar los conceptos teóricos impartidos en clase. Durante este proceso, no se muestra a los alumnos todos los procesos internos que se llevan a cabo en un compilador como el de C#, sino que las sesiones se focalizan en aquellas partes más importantes y relacionadas con los conceptos impartidos en clase.

2.1.4 Discusión

Cada una de las metodologías de enseñanza de la materia de Procesadores de Lenguaje presentadas exhibe distintas ventajas y desventajas, derivadas de los enfoques pedagógicos particulares adoptados por las mismas. Efectivamente:

- La *implementación de un procesador para un lenguaje de programación reducido*, permite a los alumnos aplicar directamente los formalismos y técnicas aprendidas en las clases teóricas en la implementación de un procesador. Además, al tratarse de un proyecto de gran envergadura que deben realizar organizándose en grupos, los alumnos tienen la oportunidad de aplicar sus conocimientos en

gestión de proyectos e ingeniería del software para mejorar la producción del software exigido y su calidad. Sin embargo, lo que suele ocurrir con esta metodología es que los alumnos se centran demasiado en la implementación del procesador, olvidándose de los aspectos relativos a su especificación. No obstante, el énfasis en una correcta especificación es fundamental para facilitar el posterior desarrollo y garantizar la corrección del procesador finalmente implementado [63].

- La realización de *pequeños proyectos de procesamiento de lenguaje* permite a los alumnos aplicar directamente las técnicas vistas en clase, a lo largo del curso, en la implementación de distintos microlenguajes. Esto ayuda a los alumnos a retener los conceptos impartidos en clase al aplicarlos, en un corto periodo de tiempo, en la práctica. Por otro lado, al igual que ocurría con la anterior, esta metodología conduce a los alumnos a centrarse en los aspectos de implementación, dejando de nuevo a un lado el fundamento teórico de éstas. Así mismo, esta estrategia no ofrece una visión completa del proceso de desarrollo global de un procesador más realista.
- Por último, en el *análisis y depuración de un procesador de lenguaje real* los alumnos pueden observar en detalle cómo se aplican las técnicas y algoritmos asociados a la creación de procesadores de lenguaje depurando un procesador de lenguaje real. Sin embargo, al presentar al alumno un procesador de lenguaje real, éste puede tener problemas para identificar los conceptos teóricos impartidos en clase y su fundamentación. Además, su rol puede llegar a ser el de mero observador, no desarrollándose adecuadamente las capacidades de especificación e implementación necesarias.

De esta forma, obsérvese que todas estas metodologías se centran fundamentalmente en aspectos de implementación. No obstante, como ya se ha indicado, la especificación de los diferentes aspectos del procesador de lenguaje es, si cabe, un aspecto mucho más importante para el correcto desarrollo de dicho programa. Por tanto, el sesgo hacia la implementación de estas metodologías tiene como consecuencia que los alumnos no sean capaces de entender en profundidad las esencias de especificación formal impartidas en clase, esencias que son la base de los procesadores implementados en los distintos proyectos propuestos en las distintas

metodologías. En este sentido, *Evaluators* trata de paliar este inconveniente ya desde una fase temprana del proceso de enseñanza / aprendizaje, convirtiéndose en una herramienta software para enseñar el modelo de proceso derivado de las gramáticas de atributos, uno de los formalismos de especificación incluidos en las materias de Procesadores de Lenguaje que mejor captura la idea de *traducción dirigida por la sintaxis* [3] en la que se basan, en última instancia, el componente de traducción de los procesadores (normalmente, el componente más complejo de construir y mantener).

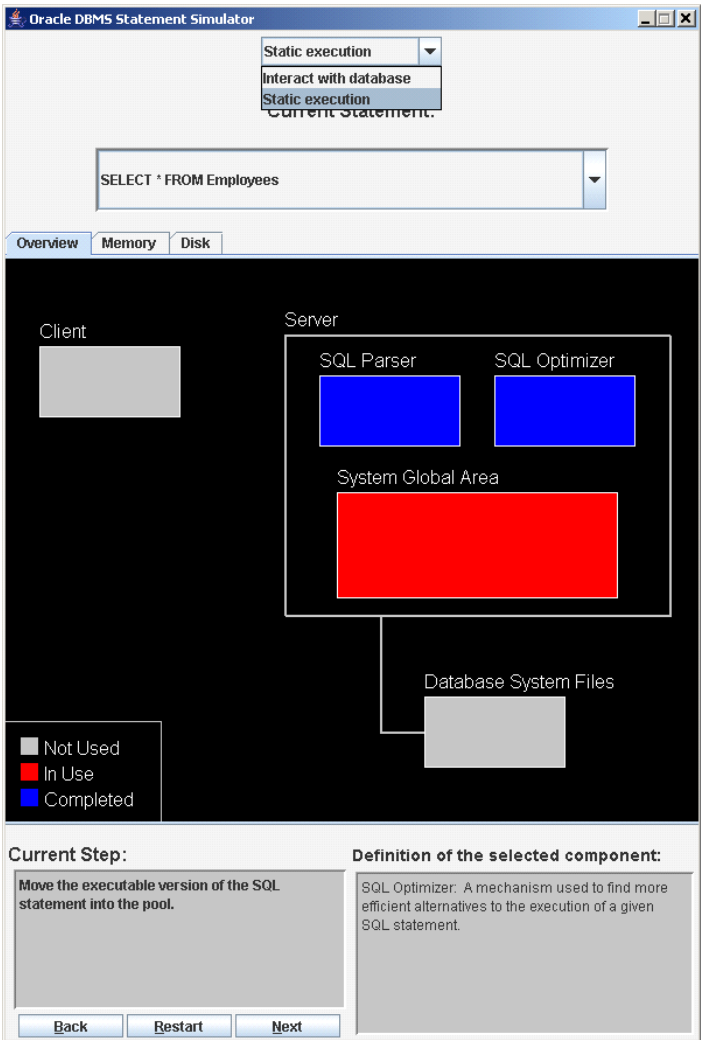
2.2 Herramientas Educativas para la Enseñanza de Ingeniería Informática

Como ya se ha indicado, el objetivo de este proyecto de investigación es proponer una herramienta informática que contribuya a mejorar la enseñanza / aprendizaje de la materia de Procesadores de Lenguaje. Ya que estas materias se enmarcan dentro del currículo de los estudios universitarios de Informática, es interesante analizar distintas herramientas software educativas que se centran en la enseñanza de determinadas materias de estos estudios universitarios. En esta sección se llevará a cabo este análisis, con el fin de estudiar distintas estrategias educativas seguidas a la hora de diseñar este tipo de herramientas. Para ello se presentan distintas herramientas software educativas que engloban aspectos tales como bases de datos, estructuras de datos o sistemas operativos. Por último, se realiza un análisis crítico, en conjunto, de todas estas herramientas.

2.2.1 Consulta a bases de datos: Query Simulation System

Query Simulation System es una herramienta planteada en el contexto de las bases de datos. Los creadores de esta herramienta pretenden ayudar a sus alumnos a entender los procesos internos que se llevan a cabo al realizar distintas consultas a una base de datos [5]. Para ello, han desarrollado un simulador que muestra, de forma simbólica, los procesos que ocurren a nivel lógico y físico durante el procesamiento de las consultas. El simulador permite realizar distintas consultas con los parámetros ya predefinidos (INSERT, UPDATE, etc.). Cuenta con las características de cualquier simulador, como *avanzar*, *paso a paso*, *pausar la ejecución*, etc. La Figura 2.1 muestra una captura del simulador.

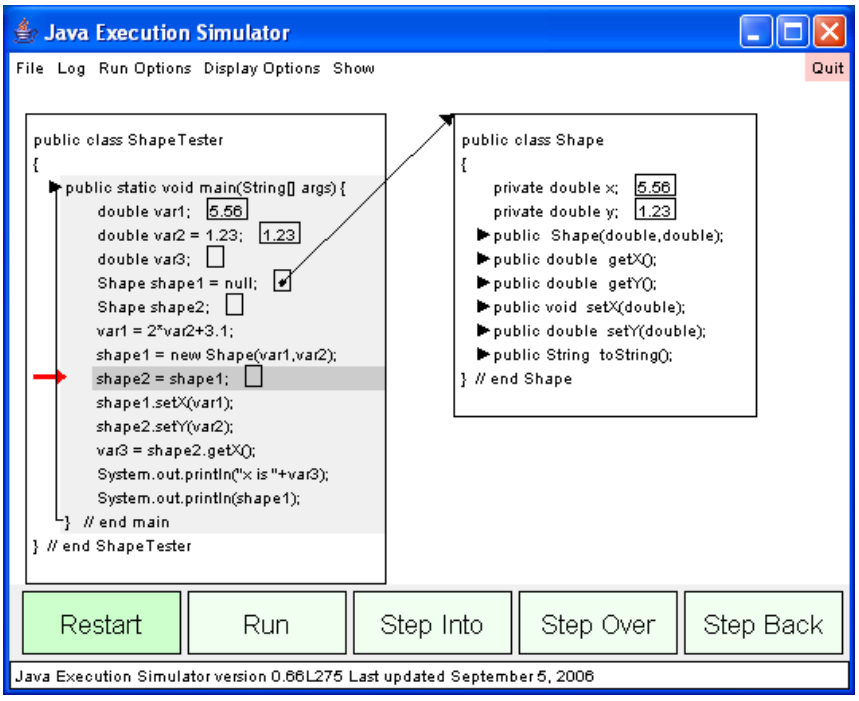
Figura 2.1. Captura del simulador del sistema de consultas de bases de datos (obtenida de [5])



2.2.2 Ejecución de programas Java: JES(Java Execution Simulator)

JES es un simulador de la ejecución de Java. El simulador soporta únicamente un subconjunto de Java (por ejemplo, no soporta booleanos, y por lo tanto, ninguna estructura de control como bucles o condicionales). A pesar de estas limitaciones, su autor defiende en [52] que es una herramienta educativa muy útil durante las primeras lecciones en programación orientada a objetos, ya que permite ver en detalle las relaciones entre las distintas clases que intervienen en la ejecución de un programa. La Figura 2.2 muestra una captura de este simulador.

Figura 2.2. Captura de JES (obtenida de [52])



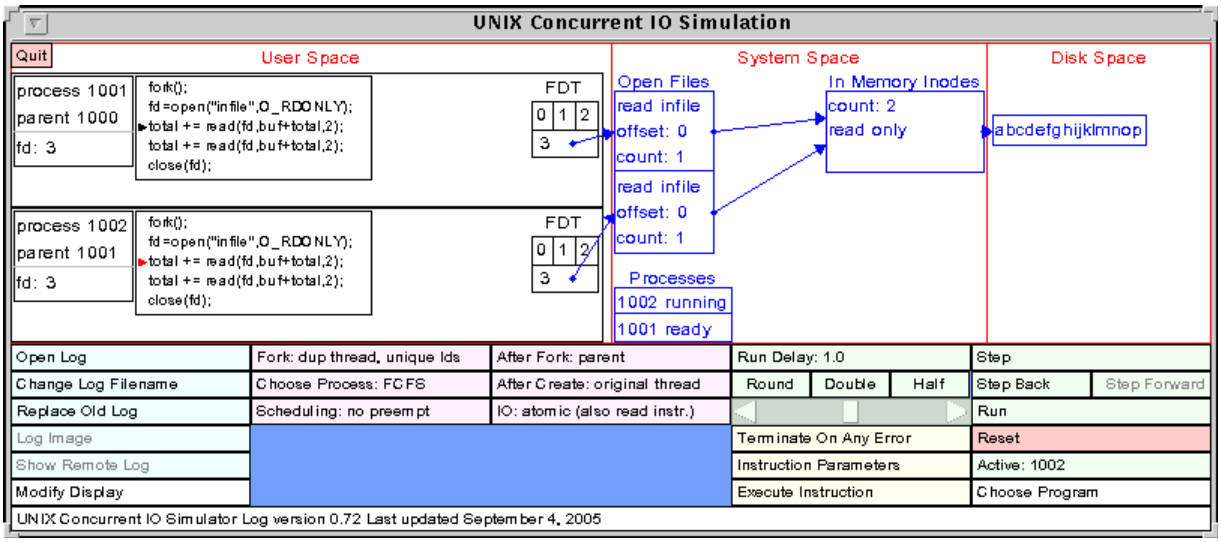
2.2.3 Simulador de Entrada/Salida Concurrente en UNIX

En [53] se describe una herramienta capaz de ejecutar programas en C e ir mostrando, en forma de tabla, la distinta información de interés para ilustrar la lectura y escritura de archivos de forma concurrente. La herramienta permite visualizar la tabla de ficheros abiertos de cada proceso o hilo, los i-nodos en memoria y la lista de procesos. También permite configurar las políticas de uso de CPU y tiene controles de avance paso a paso para ejecutar el programa introducido. Por ello, este simulador puede ser usado como un depurador. La Figura 2.3 muestra una captura de dicha herramienta con la información anteriormente descrita.

2.2.4 Simulador de Traducción de Direcciones

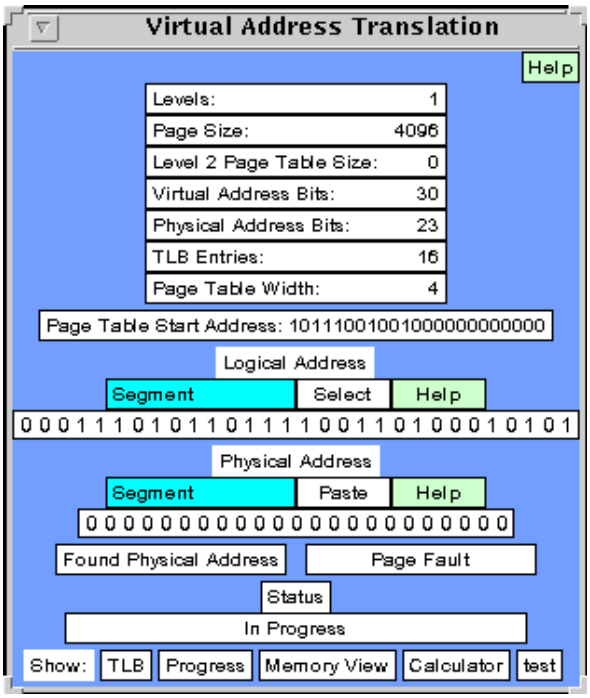
El autor de esta herramienta hace notar en [54] que, tradicionalmente, los ejemplos expuestos en clase de Sistemas Operativos para ilustrar los conceptos de traducción de direcciones suelen ser ejemplos muy simples y bastante alejados de una traducción real. Por lo tanto, presenta una herramienta muy completa donde los alumnos podrán simular una traducción real de una dirección. La Figura 2.4 muestra una captura del simulador.

Figura 2.3. Captura del simulador de entrada/salida concurrente (tomada de [53])



Una característica distintiva de este simulador es que, en todo momento, registra las acciones llevadas a cabo por el usuario, y cuando éste termina, envía la información vía e-mail al tutor para que éste pueda revisarla y proporcionar al alumno consejos y pistas personalizadas en su caso.

Figura 2.4. Captura del simulador de traducción de direcciones (tomada de [54])

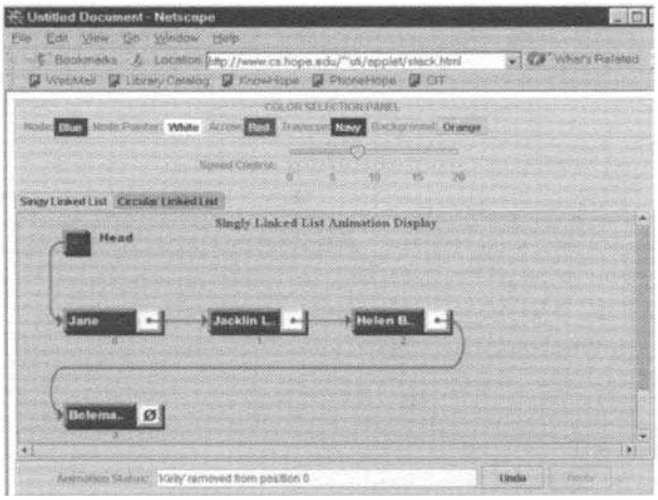


2.2.5 Animación de Listas Enlazadas en Java

En [18] se expone una biblioteca Java que pretende sustituir a las listas enlazadas predefinidas en Java para dotar a los programas creados con esta biblioteca de la posibilidad de visualizar estas listas y observar su comportamiento. Esta biblioteca proporciona una serie de interfaces que permiten manipular y visualizar estas listas de distintas formas, según se elija una implementación u otra. Las implementaciones disponibles son: listas circulares, listas doblemente enlazadas, etc. La Figura 2.5 muestra una captura de una de estas visualizaciones.

El autor de esta herramienta la propone como un complemento ideal, tanto en las clases teóricas como en los laboratorios, para ilustrar el funcionamiento de las listas enlazadas a los alumnos de una manera muy vistosa e interesante. Además, propone usar la biblioteca a los alumnos como una ayuda a la depuración de programas que dependan de esta estructura de datos.

Figura 2.5. Captura de la visualización de la lista enlazada producida por la librería (extraída de [18]).



2.2.6 Discusión

Todas las herramientas software anteriormente presentadas, a pesar de tratar diferentes temas, permiten enriquecer la experiencia educativa de los alumnos ofreciéndoles de forma visual y animada los procesos y algoritmos presentados en las clases teóricas. Además, algunas de estas herramientas, como el simulador de traductor de direcciones de memoria, permiten registrar las acciones llevadas a cabo por el usuario para que el docente pueda analizarlas y así orientar sus clases teóricas según los problemas encontrados en ese análisis. En resumen, estas características parecen ser las mínimas necesarias, pero no suficientes, para crear una

herramienta software educativa competente para las materias de Informática (en nuestro caso, las materias relacionadas con el diseño y la implementación de lenguajes informáticos).

2.3 Herramientas Educativas para la Enseñanza y el Aprendizaje de Procesadores de Lenguaje

A lo largo de los años, muchos docentes han intentado mejorar la experiencia educativa que ofrecen a sus alumnos ayudándose de distintos métodos y herramientas, tales como videos, películas, o, como se ha mostrado en la sección anterior, incluso herramientas software. Puesto que Procesadores de Lenguaje es una materia tradicional dentro de los currículos de Informática, los docentes que imparten esta materia tienen una mayor predisposición, así como el conocimiento necesario, para desarrollar herramientas software educativas que apoyen sus clases teóricas y complementen el aprendizaje de los alumnos. En esta sección se revisan algunas de estas herramientas clasificándolas en función de los siguientes tipos: simuladores de máquinas teóricas, visualizadores de los algoritmos de análisis, visualizadores de estructuras del proceso de compilación, entornos de generación de visualizadores de compiladores y herramientas genéricas de desarrollo de procesadores de lenguajes.

2.3.1 Simulación de máquinas teóricas

Las herramientas que se encargan de simular maquinas teóricas nacen de la necesidad de ofrecer a los alumnos una presentación dinámica y amena del funcionamiento de distintas maquinas teóricas y sus algoritmos. Tradicionalmente, estos conceptos se presentan utilizando una pizarra o transparencias haciendo la experiencia pobre y confusa para los alumnos, que pueden sentirse perdidos a medida que avanza la explicación. Sin embargo, las animaciones proporcionadas por este tipo de herramientas permiten a los alumnos seguir sin problemas el funcionamiento de estas máquinas y experimentar con ellas. A continuación se describen dos herramientas que simulan el funcionamiento de distintos autómatas y maquinas teóricas (autómatas finitos, autómatas con pila, máquinas de Turing, etc.).

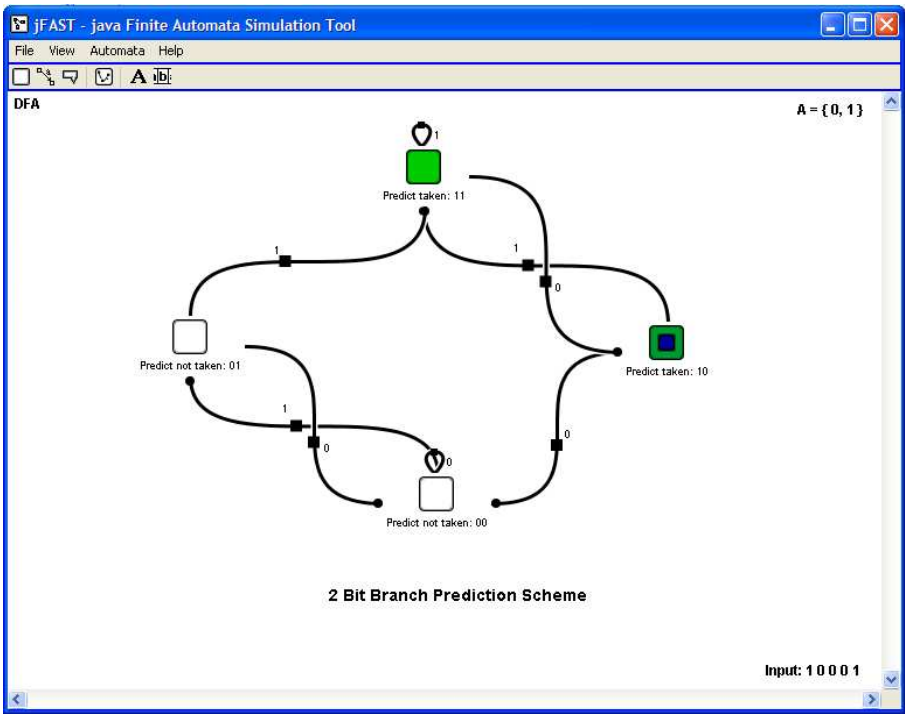
2.3.1.1 jFAST

Es un simulador desarrollado en Java que emula el comportamiento de los autómatas y de las máquinas teóricas. Ofrece una interfaz amigable y sencilla de usar que permite crear y simular distintos tipos de autómatas: autómatas finitos, autómatas de pila, máquinas de Turing,

etc. Permite, así mismo, experimentar tanto con máquinas deterministas como no deterministas. Además, permite crear los autómatas dibujándolos tal y como se haría en clase, soltando estados y conectándolos entre sí.

El software ha sido evaluado por profesores y alumnos en [76] con buenos resultados, obteniendo especial mención la sencilla forma de crear los autómatas con el simulador por parte de los alumnos. La Figura 2.6 muestra una captura de la herramienta.

Figura 2.6. Captura de jFAST (tomada de [76])

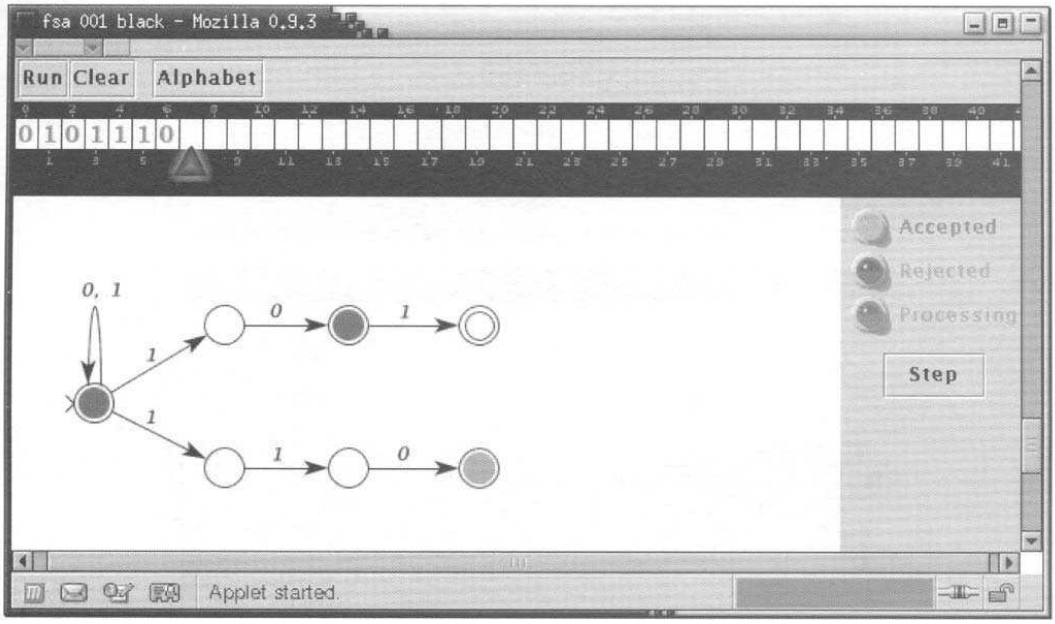


2.3.1.2 Simulador de Autómatas de Estados Finitos

En [30] se describe una herramienta que, al igual que *jFAST*, permite crear y simular autómatas de estados finitos utilizando distintos algoritmos. En comparación con la herramienta anterior, ésta presenta una característica diferenciadora muy útil, permitiendo comparar dos autómatas y extraer las diferencias entre ellos [30]. Esta característica resulta de gran utilidad para los alumnos, ya que les permite comparar los autómatas diseñados por sí mismos con la solución dada por el profesor y ver los errores que han cometido. La Figura 2.7 muestra una captura del simulador en funcionamiento.

Para probar la utilidad educativa del simulador, se llevó a cabo un estudio entre los alumnos, tal y como se indica en [31]. El estudio, sin embargo, no mostró diferencias significativas entre los alumnos que usaron el simulador y los que no.

Figura 2.7. Captura del simulador de autómatas de estados finitos (extraída de [30]).



2.3.2 Visualización de los algoritmos de análisis

Las herramientas de visualización de algoritmos de análisis ilustran el funcionamiento de distintos algoritmos de análisis sintáctico-semántico, tanto descendentes como ascendentes. Esto permite al estudiante comprender, de una forma sencilla, el funcionamiento de estos algoritmos y la finalidad de las distintas estructuras de datos implicadas en estos. Para conseguir esto, las herramientas muestran representaciones abstractas de estas estructuras y permiten a los usuarios emular el comportamiento de los algoritmos para visualizar cómo se modifican los datos de las estructuras implicadas. Para ilustrar este tipo de herramientas se presentan BURGRAM, SEFALAS y CUPV como ejemplos prototípicos.

2.3.2.1 BURGRAM

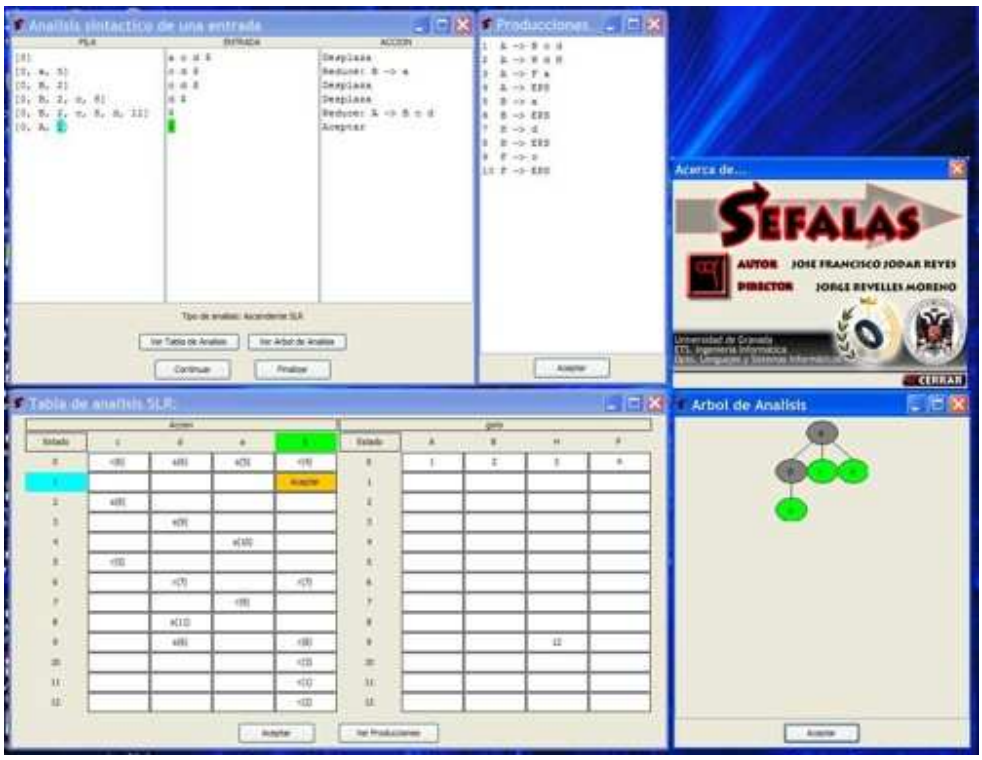
BURGRAM es una herramienta desarrollada en Java que se apoya en ANTLR [50] (herramienta de generación de procesadores de lenguaje basados en análisis descendente) y GRAPPA (paquete Java para el dibujado de grafos) [29] para mostrar el funcionamiento de distintos algoritmos de procesamiento de lenguaje, tales como los métodos de análisis sintáctico

descendentes y ascendentes [3]. Básicamente, BURGRAM es un simulador que permite emular estos algoritmos y muestra una representación de las estructuras de datos implicadas en dichos algoritmos, como pueden ser la pila de análisis. Además, tal como apunta su autor en [24], la característica más importante de la herramienta es su capacidad para registrar las acciones llevadas a cabo por el estudiante en la misma y de exportar éstas en un formato externo, como HTML o PDF, para así permitir a los alumnos añadirlas a sus apuntes.

2.3.2.2 SEFALAS

SEFALAS es un software específicamente diseñado para mostrar el funcionamiento de las fases de análisis léxico y sintáctico. Para ello cuenta con la implementación de distintos métodos de análisis, tanto léxico (con o sin anticipación) como sintáctico (analizadores ascendentes y descendentes). Las especificaciones léxicas se introducen en la herramienta en el formato LEX y las sintácticas usando el lenguaje YACC [60]. La herramienta muestra, entonces, para cada tipo de análisis, información tal como el árbol de análisis sintáctico, el autómata finito determinista, y la pila de análisis. La Figura 2.8 muestra una captura de la herramienta, tal y como ésta aparece en [35].

Figura 2.8. Captura de la herramienta SEFALAS (tomada de [35])

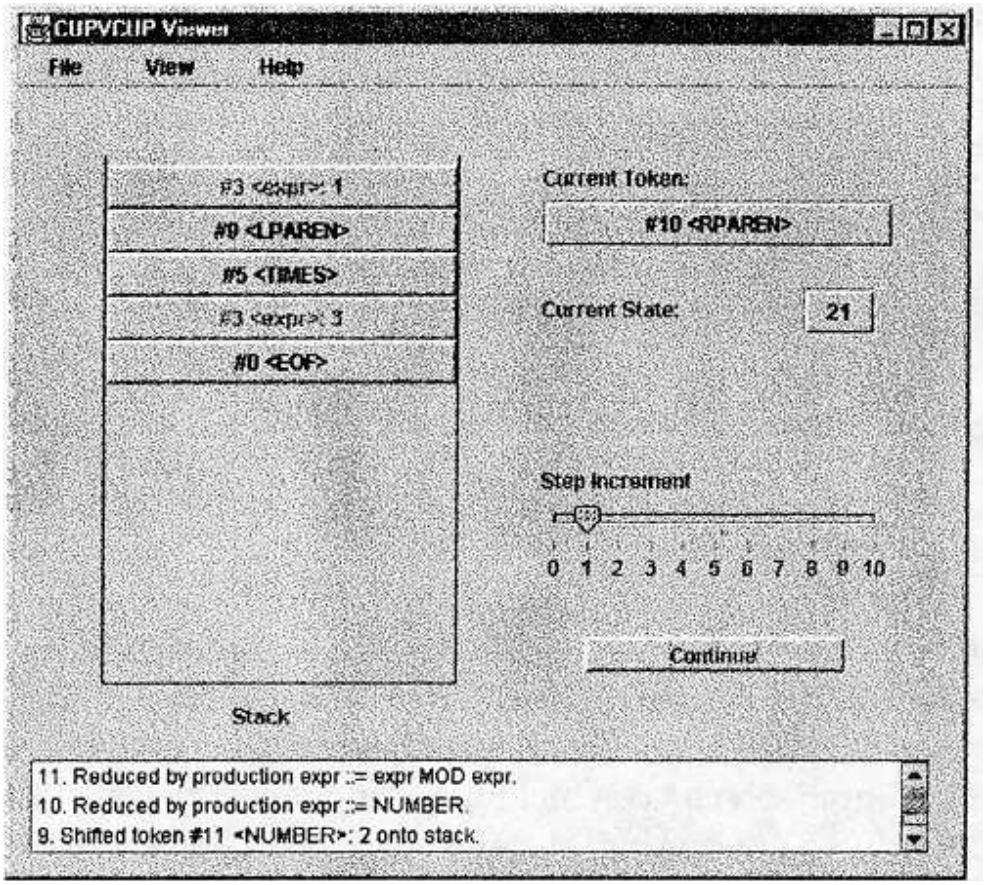


2.3.2.3 CUPV

CUPV es un depurador visual para procesadores de lenguajes especificados con la herramienta CUP (herramienta de generación de procesadores de lenguaje basados en análisis ascendente) [33]. Procesa cualquier especificación en CUP y permite visualizar la pila de análisis que se utiliza para procesar las sentencias en el lenguaje especificado, cómo se realizan las reducciones y cuál es el valor de cada posición de la pila. La Figura 2.9 muestra una captura donde se aprecia la pila de análisis que muestra CUPV.

En [36], aparte de presentarse la herramienta, se muestra un estudio realizado con alumnos de la asignatura de Procesadores de Lenguaje para evaluar la utilidad del software. En base a dicho análisis, resulta evidente que la herramienta reduce el tiempo de depuración para los alumnos, al poder visualizar el proceso de análisis llevado a cabo por los analizadores generados por CUP de una forma visual y bastante intuitiva.

Figura 2.9. Captura de CUPV (extraída de [36]).



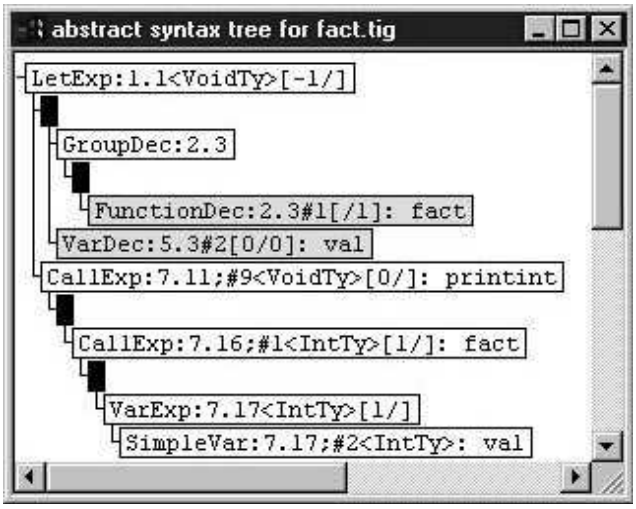
2.3.3 Visualización de las estructuras del proceso de compilación

Las herramientas de visualización de las estructuras del proceso de compilación muestran al alumno la construcción, actualización y consulta de diferentes estructuras de datos implicadas en el proceso de compilación. Estas estructuras pueden ser la tabla de símbolos o el árbol de análisis sintáctico. A continuación se ilustra este tipo de herramientas con SOTA, Tree-Viewer Library y Annotating Debugger.

2.3.3.1 SOTA

SOTA permite al alumno visualizar una tabla de símbolos de un procesador de lenguaje y simular el proceso de construcción y consulta de la misma [70]. La herramienta muestra simultáneamente el código fuente, una visualización de la tabla de símbolos y las acciones que se llevan a cabo sobre ella.

Figura 2.10. Captura de la visualización de un árbol sintáctico con Tree-Viewer Library (extraída de [71]).



Como se indica en [70], esta herramienta fue evaluada con alumnos reales. Se evaluó tanto el grado de satisfacción como la eficacia educativa de la herramienta. Se observó una gran valoración por parte de los alumnos, pero la eficacia educativa[1] resultó igual a la conseguida por el método tradicional de enseñanza de este concepto. Por otro lado, se observó en el estudio que la eficiencia educativa[2] aumentaba con los estudiantes que usaron la herramienta.

[1] La eficacia educativa se refiere a *lo bien que aprende* un alumno.

[2] La eficiencia educativa se refiere a *lo rápido que aprende* un alumno. De esta forma, un método de aprendizaje eficaz no tiene porque ser eficiente, y viceversa.

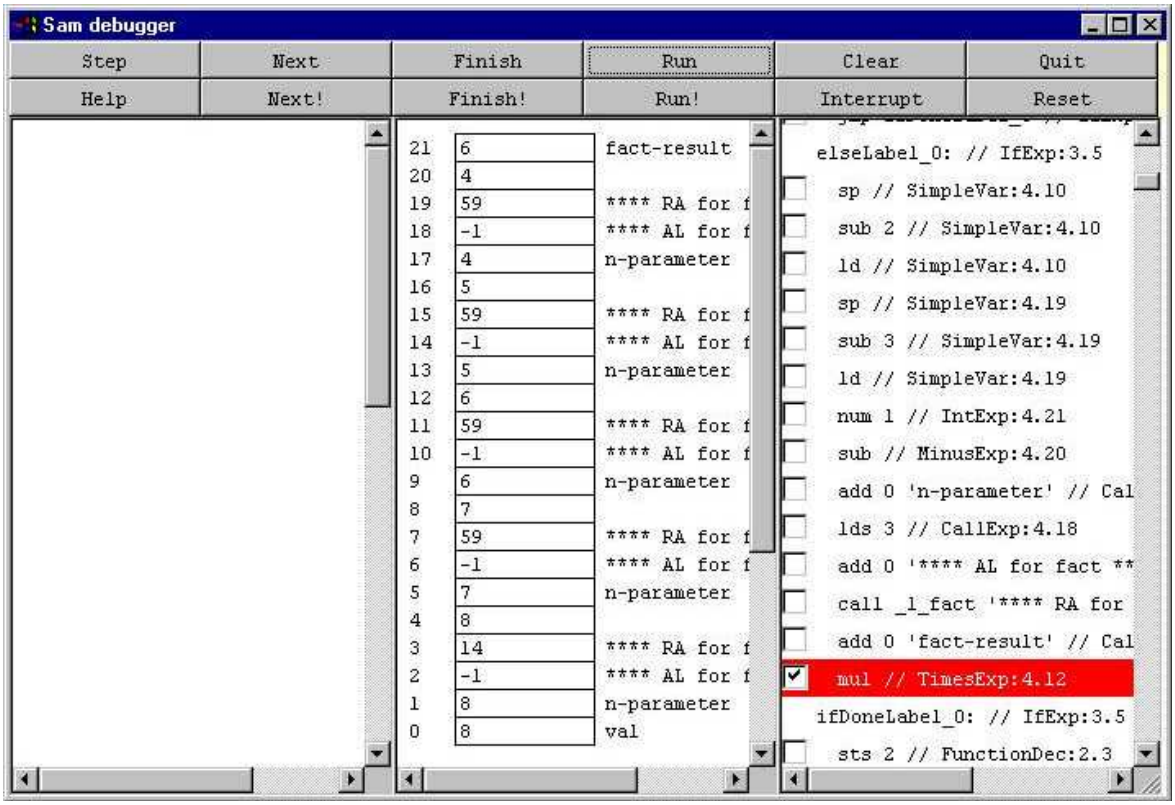
2.3.3.2 Tree-Viewer Library

Tree-Viewer Library es una biblioteca java orientada a ayudar a los alumnos a entender el proceso dirigido por sintaxis para la construcción de compiladores [71]. Los alumnos deben incluir esta librería en sus proyectos para que estos generen automáticamente visualizaciones de los árboles sintácticos (véase la Figura 2.10).

Esta herramienta sirve para ayudar tanto a docentes como a alumnos a depurar sus compiladores y entender, de una forma visual y bastante atractiva, cuáles pueden ser los errores que están cometiendo.

2.3.3.3 Annotating Debugger

Figura 2.11. Captura de la visualización de la maquina pila generada por Annotating Debugger (extraída de [71]).



Annotating Debugger [71] es un depurador que permite visualizar la máquina virtual basada en pila que ejecutará el código maquina generado por el procesador del lenguaje. La gran ventaja que proporciona es la de incluir anotaciones en el código maquina generado por el compilador que se visualizarán en la maquina basada en pila. Esto último ayuda a entender el

proceso que se lleva a cabo dentro de la máquina, ya que las anotaciones le indican al alumno a qué corresponde cada valor de la pila (véase la Figura 2.11). Esto permite a los alumnos entender la relación entre el código maquina generado y el código del procesador del lenguaje que han programado. Además, opinan los autores, Annotation Debugger es una herramienta muy útil a la hora de depurar sus compiladores.

2.3.4 Entornos de generación de visualizadores de compiladores

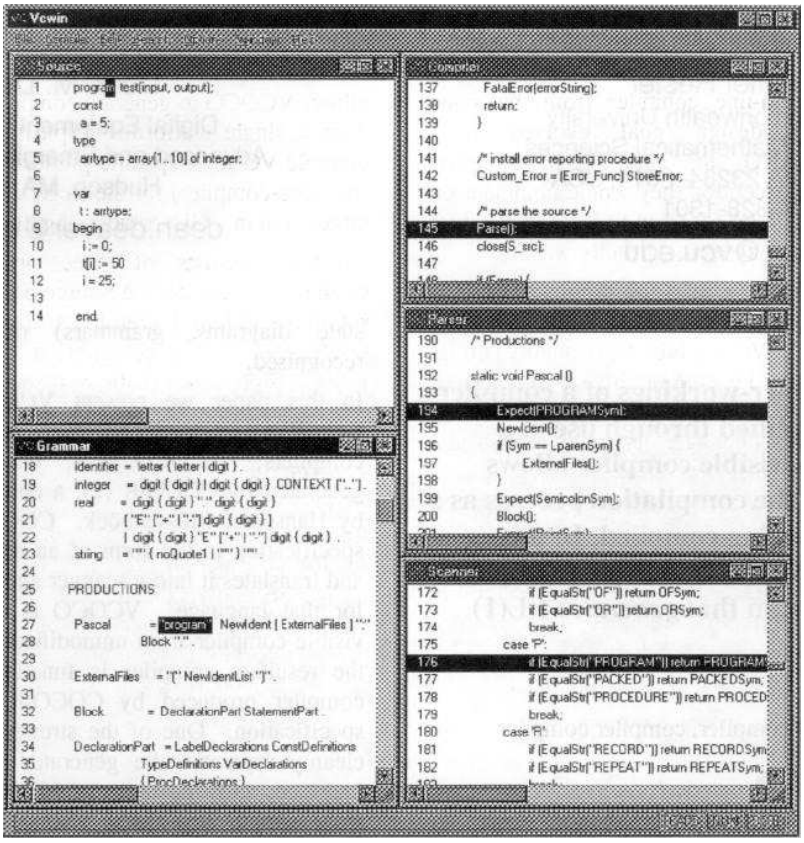
Estas herramientas permiten al usuario crear procesadores de lenguajes basándose en las especificaciones de estos lenguajes. Además proporcionan una visualización enriquecida de los procesos que se llevan a cabo para procesar las sentencias que consumen los compiladores creados. Con esto se persigue que el estudiante sea capaz de comprender en profundidad el funcionamiento de los procesadores de lenguaje que él mismo especifica y ofrecerle, al mismo tiempo, un depurador enriquecido y orientado a sus necesidades para ejecutar los procesadores de lenguaje que él especifique. A modo de ejemplos, se presentan las herramientas VCOCO y PAG.

2.3.4.1 VCOCO

VCOCO es una herramienta de visualización que puede ser entendida como un depurador muy sofisticado para el lenguaje de creación de compiladores COCO. COCO, similar a JavaCC o YACC, permite especificar lenguajes mediantes gramáticas EBNF anotadas con acciones semánticas y construir su correspondiente procesador.

Esta herramienta, que se describe en [16], permite al usuario visualizar simultáneamente la gramática, la sentencia, el código maquina generado y el código fuente del programa que se está compilando. El depurador va resaltando las partes implicadas, simultáneamente, en cada uno de estos objetos a medida que la ejecución avanza (véase la Figura 2.12). Esto resulta de utilidad a los alumnos para comprender el funcionamiento, como unidad, de todas las partes que conforman un procesador de lenguaje.

Figura 2.12. Captura de VCOCO (extraída de [16]).



2.3.4.2 PAG

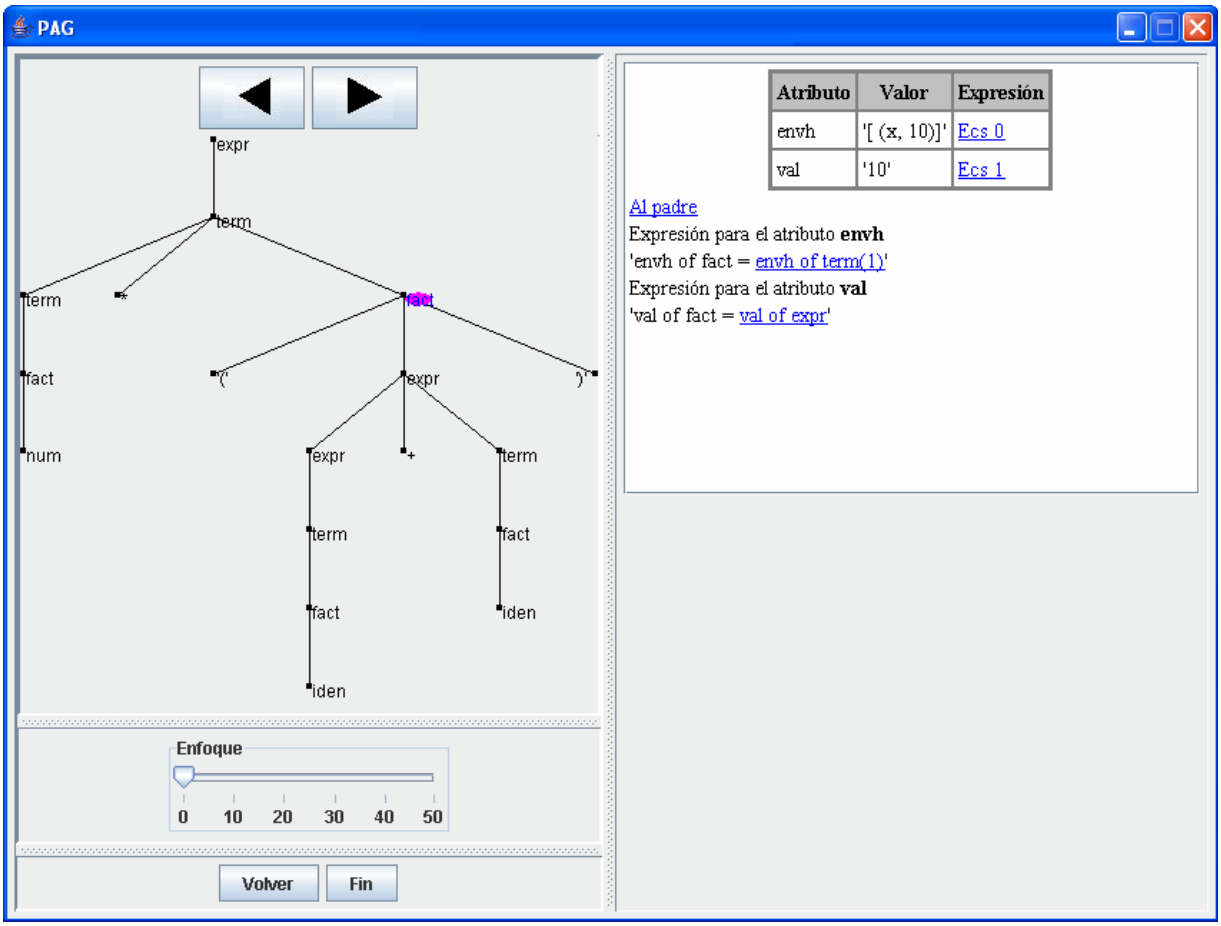
PAG (*Prototyping with Attribute Grammars*) es un entorno de desarrollo que permite desarrollar prototipos de procesadores de lenguajes a partir de especificaciones basadas en gramáticas de atributos (véase la sección 2.5). Las especificaciones de los lenguajes se escriben en una forma muy similar a la empleada en la especificación de las gramáticas de atributos, mediante una sintaxis embebida en el lenguaje Prolog [69][3]. El usuario debe especificar, aparte de la gramática (o fragmentos de gramáticas) que compone(n) el lenguaje, las funciones semánticas que se utilicen en la gramática de atributos. El usuario podrá ejecutar los prototipos de procesadores de lenguajes creados analizando distintas sentencias y obtendrá como resultado todos los posibles árboles sintácticos decorados (si la sentencia pertenece al lenguaje) que podrá

[3] Un lenguaje embebido es aquel cuyas sentencias se escriben en forma de expresiones de un lenguaje huésped [23]. De esta forma, las sentencias del lenguaje son también expresiones válidas en dicho lenguaje huésped. Prolog, dado que permite definir nuevos operadores proporcionando información sobre su nivel de prioridad y asociatividad, es especialmente adecuado para embeber lenguajes [44].

visualizar y explorar para comprobar el valor de los atributos y cómo se han computado estos según se ha especificado en la gramática que define el lenguaje. La Figura 2.13 muestra una captura de un árbol decorado generado por PAG, tomada de [63].

En [63] se describe el uso educativo del entorno en dos disciplinas distintas: en Informática dentro de la materia de Procesadores de Lenguajes y en Filología dentro de la materia de Lingüística Computacional.

Figura 2.13 Captura de la herramienta PAG (tomada de [63])



2.3.5 Herramientas genéricas de desarrollo de procesadores de lenguaje

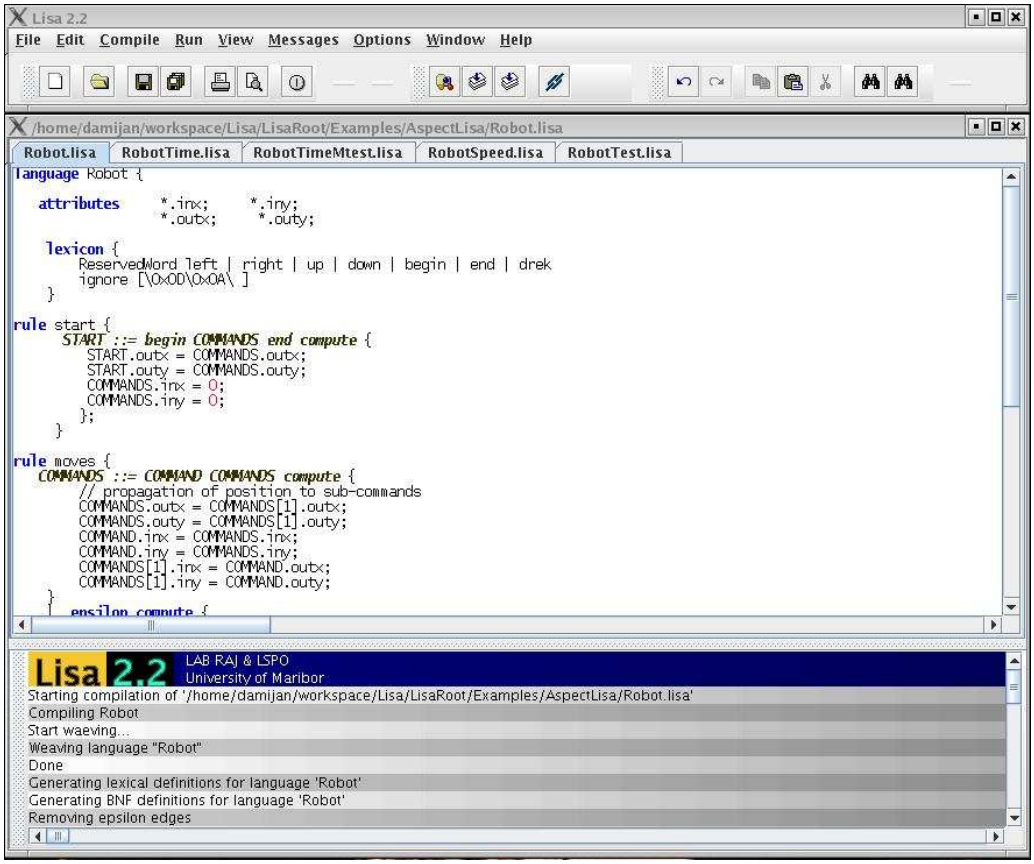
Las herramientas genéricas de desarrollo de procesadores de lenguaje, también conocidas como *compiladores de compiladores*, son herramientas software que permiten al desarrollador crear procesadores de lenguaje de una forma muy sencilla, a partir de especificaciones de alto nivel de los mismos. Efectivamente, estas herramientas procesan una especificación del lenguaje para generar automáticamente un programa que procese el lenguaje especificado. Este tipo de

software es ampliamente empleado para el desarrollo profesional de compiladores, pero también puede ser utilizado como herramientas educativas para cubrir algunos de los conceptos impartidos en la materia de Procesadores de Lenguaje. A continuación se describen dos trabajos donde se emplean estas herramientas con un fin educativo.

2.3.5.1 LISA

LISA permite mostrar a los alumnos las fases de análisis léxico, sintáctico y semántico de manera visual y amigable. Los alumnos deben especificar el lenguaje mediante expresiones regulares para los aspectos léxicos, y mediante gramáticas de atributos para el resto de los aspectos. Con esta información, la herramienta es capaz de generar el procesador (analizador léxico, y traductor basado en el modelo de evaluación semántica de las gramáticas de atributos). La Figura 2.14 muestra una captura de la herramienta.

Figura 2.14. Captura de la herramienta LISA



En [45] se realiza un estudio del grado de satisfacción de los estudiantes con la herramienta, y se observa que los alumnos se sienten cómodos usando la herramienta y que les

resulta útil a la hora de experimentar con distintos algoritmos para cualquiera de las tres fases mencionadas anteriormente. Además, el trabajo recoge también opiniones de algunos docentes que han usado la herramienta, entre las que destaca que los alumnos parecen más interesados en la asignatura gracias a la herramienta.

2.3.5.2 ANTLRWorks

ANTLR [50] es una herramienta que, al igual que JavaCC o YaCC, permite crear procesadores de lenguaje a partir de la especificación en forma de gramática. En [14] se describe una serie de herramientas que están orientadas a enriquecer las ya ofrecidas por ANTLR. ANTLRWorks está compuesto por un editor de gramáticas, un intérprete para asistir la fase de prototipado de la gramática y un depurador independiente del idioma para identificar posibles errores en la gramática. La mayor característica de esta herramienta es que permite al usuario identificar posibles problemas de ambigüedad en el lenguaje definido y solucionarlos. Todas estas funcionalidades pueden servir de utilidad a los estudiantes a la hora de identificar posibles errores cometidos en la realización de los compiladores exigidos en clase. La Figura 2.15 muestra una captura de la herramienta ANTLRWorks, mientras ésta se usa para identificar un problema de ambigüedad en la gramática de entrada.

2.3.6 Discusión

A lo largo de las subsecciones anteriores se han descrito diferentes tipos de herramientas software educativas dirigidas a la enseñanza de materias relacionadas con los procesadores de lenguaje, ilustrándolas con varios ejemplos de estos tipos:

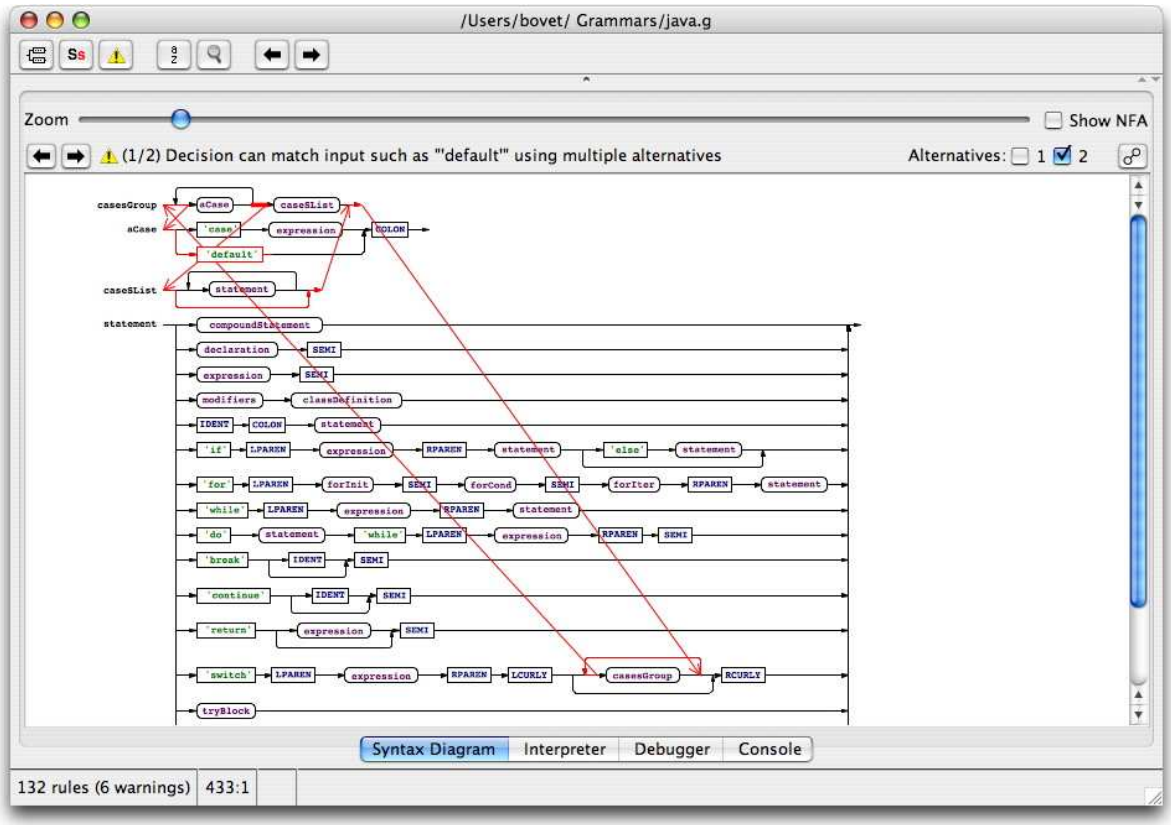
- En el primer grupo, los simuladores de las maquinas teóricas muestran la importancia de mostrar de forma visual y animada distintos conceptos teóricos fundamentales del procesamiento de lenguaje para ayudar a los alumnos a entender estos fundamentos. Su principal ventaja es incidir en los modelos teóricos subyacentes al procesamiento de lenguaje. No obstante, dichos modelos se abordan desde un punto de vista excesivamente abstracto, que, en realidad, no es el usado por los modelos más concretos utilizados durante el procesamiento. Así por ejemplo, mientras que los analizadores sintácticos pueden verse como autómatas a pila, en realidad el modelo teórico en sí no es el finalmente utilizado

para soportar el análisis sintáctico, sino una especialización de dicho modelo (por ejemplo, un analizador ascendente dirigido por tablas LR).

- El grupo de visualización de los algoritmos de análisis permite a los alumnos explorar y experimentar en profundidad con el funcionamiento los distintos algoritmos de análisis sintáctico de forma visual. No obstante, estas herramientas abordan únicamente un aspecto del procesamiento de lenguaje. Igualmente importante es, sin embargo, observar cómo este aspecto se acopla con otros durante dicho procesamiento (por ejemplo, en la traducción descendente o ascendente en una pasada). Dichas herramientas ofrecen, por tanto, una visión parcial.
- Las herramientas de visualización de las estructuras del proceso de compilación muestran a los estudiantes el funcionamiento y uso de determinadas estructuras de datos empleadas en el procesamiento de un lenguaje. Esto les proporciona un profundo conocimiento del funcionamiento interno de un procesador de lenguaje, pero sólo a nivel de implementación, dejando de lado los aspectos teóricos.
- Los entornos de generación de visualizadores de compiladores son herramientas software específicamente diseñadas para ayudar a los alumnos a crear y depurar procesadores de lenguajes. Estos entornos contienen diferentes herramientas, habitualmente visuales, que permiten a los alumnos entender de una forma más completa, con un mínimo esfuerzo, el funcionamiento de los procesadores de lenguaje que han especificado. De éstas, algunas adoptan un enfoque *entrada – salida*, en la que las distintas fases del procesamiento se ven como *cajas negras*, lo que no ayuda demasiado a la hora de que el alumno entienda cómo funcionan realmente dichas fases. Otras, como PAG, permiten al alumno especificar gramáticas para problemas de procesamiento de lenguaje genéricos. La principal ventaja de este segundo tipo de herramientas es que se focalizan en la especificación, lo que las convierten en buenos antecedentes para nuestra propuesta. No obstante, no debe olvidarse tampoco el peligro que supone el abstraerse totalmente de detalles de implementación, por lo que este tipo de herramientas debe coordinarse con otras más orientadas a dichos aspectos de implementación.

- Por último, el uso de herramientas genéricas de desarrollo de procesadores de lenguaje permiten al alumno comprobar la potencia de una herramienta software real y las herramientas adicionales de depuración les permiten entender en profundidad el funcionamiento de los algoritmos de procesamiento implementados por éstas. La principal desventaja de las mismas reside, no obstante, en el hecho de ser herramientas de desarrollo, y no herramientas educativas: por una parte su uso puede resultar complicado para el alumno, sobre todo en las fases iniciales de aprendizaje, y, por otra, estas herramientas pueden carecer de la necesaria fundamentación pedagógica en la que ha de basarse necesariamente toda herramienta utilizada con propósitos educativos.

Figura 2.15. Captura de la herramienta ANTLRWorks identificando un problema de ambigüedad (tomada de [14])



Para finalizar, indicar que muchas de estas herramientas software, se centran casi con exclusividad en la fase de implementación. Esto es cierto incluso en el caso de las herramientas

de simulación de máquinas teóricas, ya que dichas máquinas son modelos muy abstractos de componentes finales de los procesadores. De hecho, de todas las herramientas analizadas, únicamente PAG y LISA se basan en un formalismo de especificación estándar en descripción de procesadores dirigidos por sintaxis: las gramáticas de atributos. De éstas, únicamente PAG tiene un propósito primariamente educativo. No obstante, el principal problema de PAG es su orientación a alumnos que ya manejan con cierta soltura el formalismo de especificación. Se observa, por tanto, cierta carencia en herramientas que permitan adquirir conceptos básicos de procesamiento dirigido por sintaxis basado en especificaciones de alto nivel, y en etapas iniciales del aprendizaje. *Evaluators* pretende suplir esta laguna para ayudar en el aprendizaje de estos primeros conceptos en los que se apoyará el resto del proceso de aprendizaje de la materia.

2.4 Videojuegos Educativos para la Enseñanza de la Informática

Evaluators es una herramienta que promueve el uso de videojuegos para aprender conceptos básicos sobre procesadores de lenguaje. En esta sección se revisarán herramientas similares.

El uso de videojuegos educativos constituye una tendencia creciente en el campo del e-Learning [6][25][26]. El uso de juegos para enseñar no es, en modo alguno, una idea nueva, ya que durante muchos años se han empleado juegos (no solo videojuegos) para enseñar distintos conceptos y fomentar la adquisición de distintas capacidades y competencias. Efectivamente, los videojuegos permiten sumergir al usuario en un mundo virtual diseñado específicamente para enseñar una serie de procesos y técnicas muy cercanas a las que podría encontrarse en la vida real [61], y cuentan con cuatro características fundamentales que los convierten en una herramienta única de aprendizaje [25]:

- Los videojuegos son divertidos. Además, esta característica no es únicamente aplicable a una población joven. Efectivamente, la edad media de los jugadores comprende los 30 años, y cerca de un 43% de los consumidores de este tipo de software son mujeres. Por lo tanto, se puede concluir que los videojuegos pueden llamar la atención de gran parte de la población.
- Los videojuegos son inmersivos. De esta forma, estos programas permiten al usuario introducirse en un mundo virtual e interactuar con él. Esto permite a los diseñadores crear mundos virtuales en los que el usuario pueda aprender

experiencias muy cercanas a los objetivos educativos con los que se diseñaron estos mundos.

- Los videojuegos estimulan la cooperación y la competitividad. Esto ayuda, desde un punto de vista psicológico, al aprendizaje colaborativo sin ser necesario la participación de compañeros reales (utilización de NPCs: *Non-Player Characters*). Además la competitividad es siempre un estimulo muy poderoso.
- Los videojuegos estimulan la creación de comunidades de usuarios. Estas comunidades son lugares donde los usuarios comparten sus conocimientos sobre un videojuego con otros usuarios, donde pueden discutir, donde pueden aprender a ser mejores jugadores gracias a guías hechas por otros usuarios. Estas comunidades son una continuación de la actividad del videojuego fuera del videojuego. Nótese que esta característica es difícil de encontrar en las escuelas convencionales, ya que los estudiantes promedio no suelen tratar temas académicos fuera de la escuela.

Por todo esto, los videojuegos educativos son, potencialmente, una poderosa herramienta para la educación. De esta forma, la educación en Informática se ha hecho eco también de estos usos educativos de los videojuegos. En esta sección se analizan diferentes videojuegos educativos orientados a facilitar la enseñanza y el aprendizaje de materias propias de la informática.

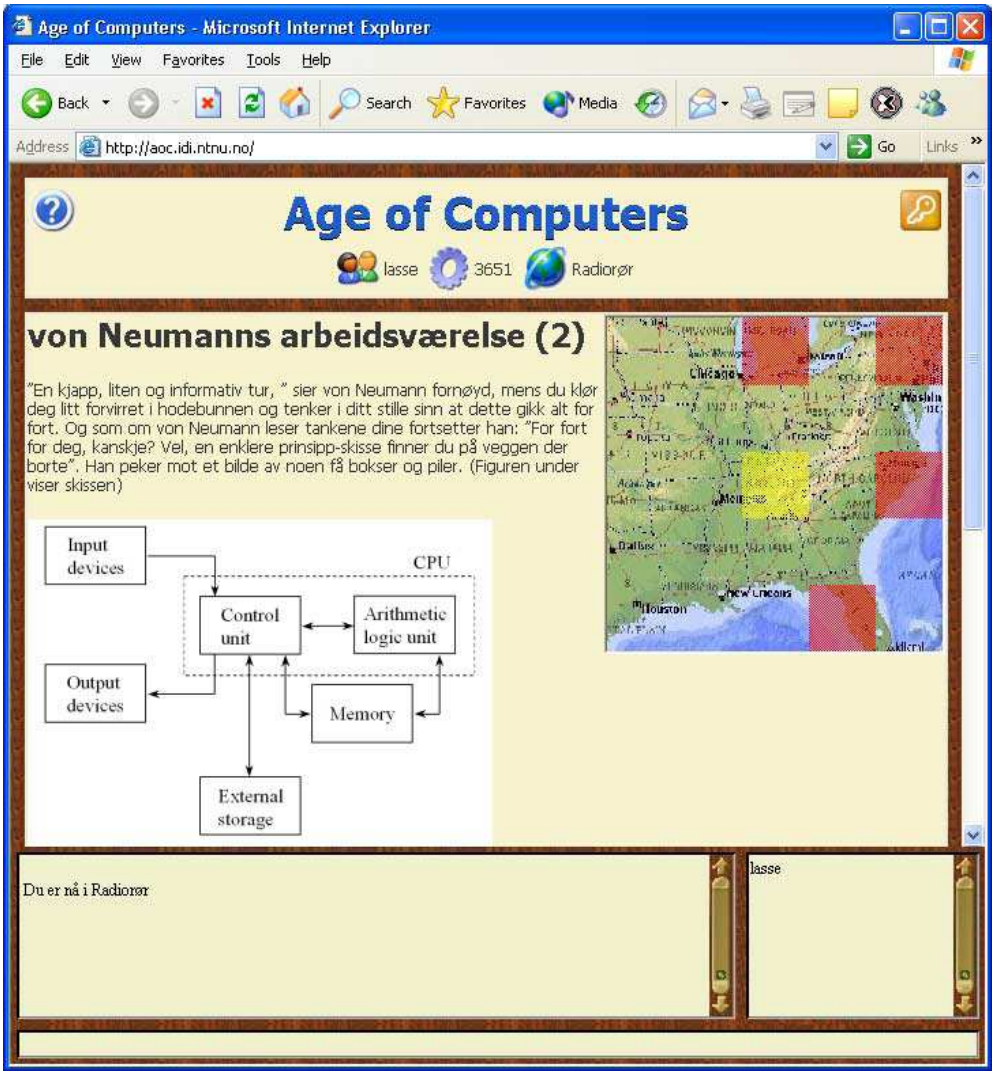
2.4.1 Fundamentos de Computadores: Age of Computers

Age of Computers [47] es un videojuego educativo desarrollado por el *Instituto Noruego de Tecnología y Ciencia* orientado a los alumnos de la asignatura de un primer curso de Fundamentos de Computadores. El videojuego sitúa al usuario en distintas épocas de la informática (por ejemplo, la edad de los ordenadores mecánicos, la de las válvulas de vacío o la de los circuitos integrados). En cada una de estas épocas el alumno tendrá que resolver distintos problemas para ir aumentando su puntación e ir avanzando a la siguiente época. La Figura 2.16 muestra una captura del videojuego.

Además, el videojuego cuenta con un chat donde el alumno puede comunicarse con el resto de usuarios del juego que se encuentren en la misma época en la que se encuentra él. Con esto se incentiva el aprendizaje colaborativo.

Por último, en [47] se describe un estudio realizado para medir la satisfacción de los alumnos. Los resultados son prometedores, obteniendo buenos resultados en prácticamente todos los aspectos del videojuego. Aun así, se observa que los alumnos piden mejores formas de colaboración entre los usuarios a la hora de resolver los problemas propuestos por el videojuego.

Figura 2.16. Captura del videojuego *Age of Computers* (tomada de [47])

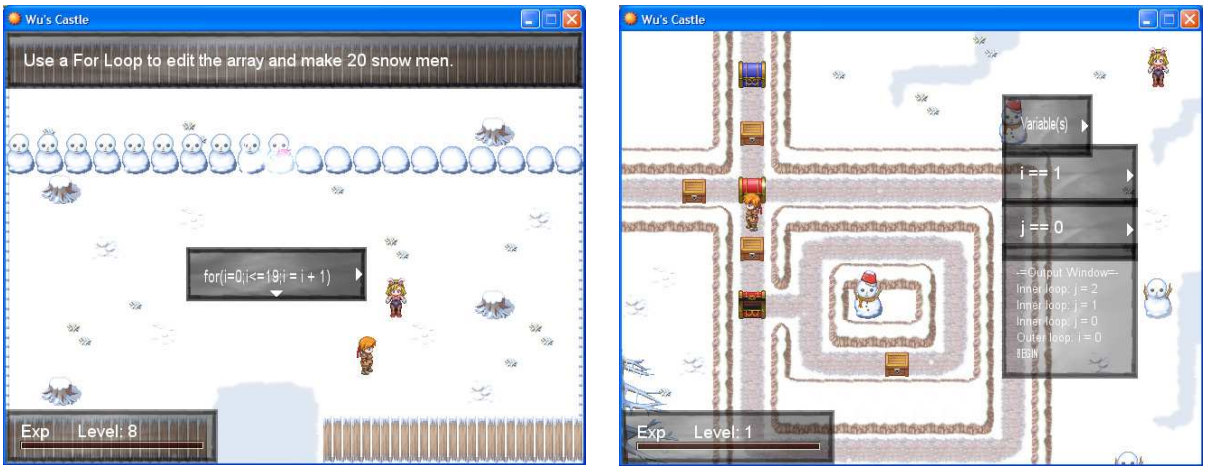


2.4.2 Programación básica: Wu's Castle

Wu's Castle es un juego simple en dos dimensiones, que está diseñado para enseñar a los alumnos de los primeros cursos de la carrera el funcionamiento de los bucles [19]. Para ello dispone de diferentes niveles que se clasifican en dos tipos o modos de juego. En el primero, el usuario tendrá que configurar un bucle "for" para recorrer una fila, o varias, de muñecos de nieve

y modificarlos según cuál sea el objetivo propuesto. El otro modo de juego consiste en mover al personaje que representa al usuario para simular el recorrido que se realiza según el bucle especificado en el nivel. La Figura 2.17 muestra capturas de los dos tipos de niveles a los que se puede enfrentar el usuario.

Figura 2.17. Capturas de las distintas modalidades de juego en *Wu's Castle* (tomadas de [19])



Para evaluar, tanto la satisfacción de los usuarios, como la efectividad educativa, en [20] se realizó un estudio con los alumnos. Se dividió a los alumnos en tres grupos: los alumnos que no realizaron los tests ni jugaron al videojuego, los que realizaron los tests pero no jugaron al videojuego y los que realizaron los tests y jugaron al videojuego. Tras evaluar los tests y los exámenes finales de la asignatura se observa que los alumnos que jugaron al videojuego obtuvieron una puntuación mejor que los que no, con una diferencia de cerca de siete puntos sobre cien en comparación con los que no jugaron al videojuego.

2.4.3 Entrada/Salida en Nintendo® DS.

En [40] se presenta una estrategia educativa basada en juegos aplicada a la enseñanza de la entrada/salida en los computadores. En esta estrategia se utiliza la videoconsola Nintendo® DS para entrenar a los alumnos en estos conceptos. La estrategia se divide en tres fases:

- En primer lugar, y en parejas de dos, se les va presentado simultáneamente a los alumnos una serie de cuestiones teóricas con diferentes alternativas para responder. Cada componente de la pareja responde a las mismas preguntas

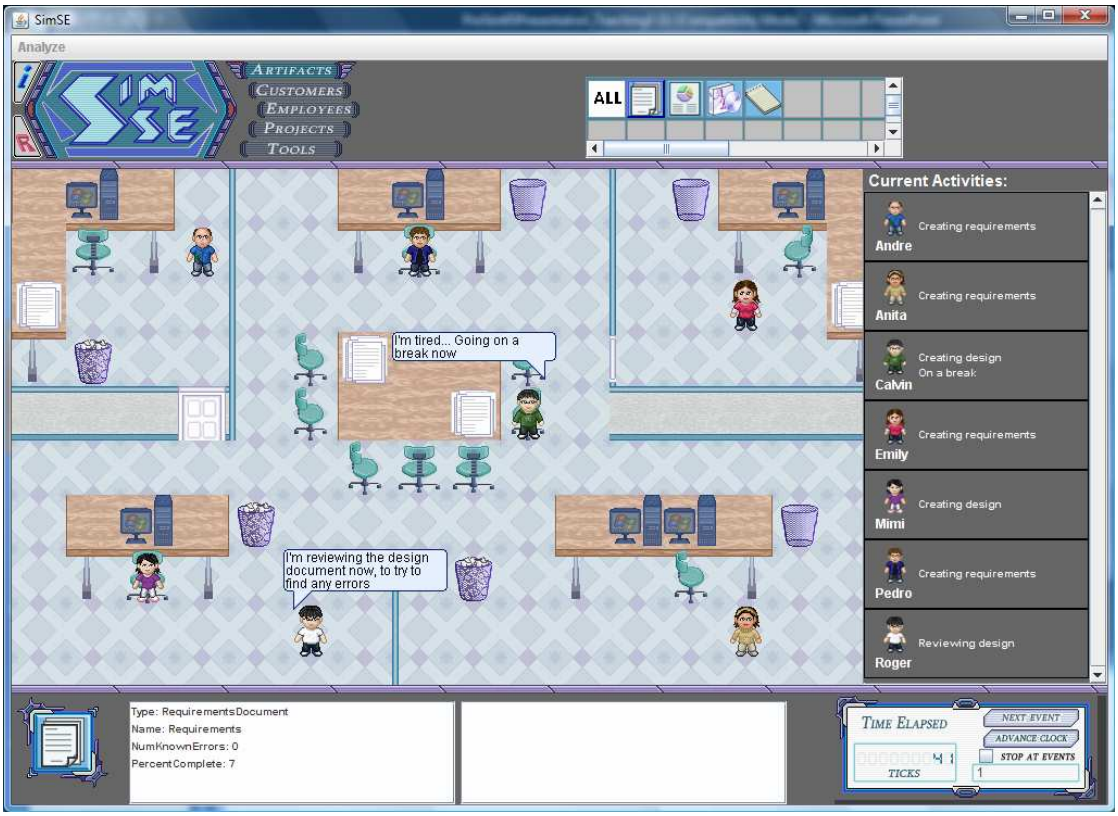
simultáneamente. Si los dos aciertan la pregunta se llevan toda la puntuación. Si alguno de ellos falla, ninguno se llevará la puntuación entera. Esto fomenta el aprendizaje colaborativo, al depender del compañero.

- Tras superar el cuestionario, el videojuego les obliga a construir un autómata de entrada/salida de forma colaborativa.
- Por último, de forma individual, cada alumno responderá a las mismas preguntas del primer test siguiendo el sistema Lietner [42].

2.4.4 Técnicas de gestión de proyectos: SimSE

SimSE es un simulador que sitúa al usuario al frente de un grupo de desarrollo de software como gestor de los proyectos que dicho grupo tenga que realizar. El estudiante debe seguir las buenas costumbres de distintas metodologías de gestión de proyectos y así ganar más puntos en el videojuego. Al final de la partida, se le presenta al usuario un informe completo de su actividad con algunas pistas y consejos sobre las malas decisiones que tomó durante la partida. La Figura 2.18 muestra una captura de SimSE.

Figura 2.18. Captura del simulador SimSE (tomada de [48])



En [48] se realiza un completo estudio de la eficacia educativa de la herramienta en diferentes universidades, observándose que la herramienta presenta buenos resultados en general. En particular, se observa que es necesario instruir correctamente a los alumnos en el uso de la herramienta ya que, sin esa ayuda, el videojuego no resulta de utilidad.

2.4.5 Máquina Virtual de Java: JV²M

JV²M es un sistema educativo basado en videojuegos que pretende enseñar el funcionamiento interno de la máquina virtual de java [27]. El objetivo final es que los alumnos comprendan la ejecución interna de los programas Java, y, de esta forma, comprendan indirectamente la traducción de estos programas a *bytecode*. Para ello presenta una ciudad donde los edificios y los objetos dentro de estos representan cada una de las partes de la máquina virtual, y el usuario deberá ejecutar un programa Java usando los elementos y edificios que encuentre en esa ciudad. La Figura 2.19 muestra algunas capturas de este juego.

Figura 2.19. Capturas de JV²M (tomadas de [27])

En JV²M, el usuario maneja un avatar que se mueve dentro de un mundo que representa la maquina virtual de Java. Un aspecto bastante novedoso que presenta este videojuego es la inclusión de un *tutor inteligente* [77] integrado dentro del videojuego, un sistema inteligente representado como un acompañante del avatar al que el usuario puede hacer preguntas o incluso pedirle que solucione el problema propuesto. Otro aspecto importante de este sistema es la arquitectura interna del juego, ya que separa de forma muy clara los contenidos a enseñar con la forma de presentarlos. De esta forma, JV²M puede configurarse a partir de descripciones de dichos problemas, junto con el resto de información necesaria para asistir al proceso de resolución por parte del alumno [27]. En *Evaluators* se sigue, de esta forma, un enfoque similar,

permitiendo la configuración del juego a partir de problemas de propagación de atributos en árboles de análisis sintáctico.

2.4.6 Discusión

A lo largo de las subsecciones anteriores se han descrito distintos videojuegos educativos dirigidos a enseñar diferentes aspectos de la informática. El principal factor que define estas herramientas es que resultan amenas a los alumnos y les invita a utilizarlos, y con ello a aprender. Un claro ejemplo, de los presentados, que explota este aspecto es el videojuego que se utiliza en una Nintendo® DS, ya que este hecho puede propiciar un gran interés por parte de los alumnos. Otro aspecto muy importante es la colaboración que puede darse en este tipo de herramientas educativas, tanto dentro (e.g., *Age of Computers*) como fuera de ellas, lo que mejora el aprendizaje de los alumnos. Por último, estos videojuegos cuentan con la característica de proporcionar de forma instantánea pistas a los usuarios e incluso la solución completa del problema propuesto, con lo que el usuario es capaz de aprender inmediatamente de los fallos que haya podido cometer (como en SimSE).

A modo de conclusión, indicar que algunas de las ideas extraídas de estos videojuegos educativos serán incorporadas a *Evaluators* como por ejemplo, una versión muy sencilla de la presentación de pistas y consejos al usuario incluida en JV^2M.

2.5 Gramáticas de Atributos

Evaluators persigue enseñar el funcionamiento básico del formalismo de las gramáticas de atributos. Este formalismo, propuesto por D. Knuth a finales de los sesenta, presenta un mecanismo para añadir semántica a las gramáticas incontextuales [3][38][39][49]. La Figura 2.20 muestra un ejemplo de gramática de atributos. A continuación se explicará brevemente este formalismo explicando los principales conceptos del mismo, a fin de establecer los conceptos mínimos necesarios para comprender el propósito y funcionamiento de *Evaluators*.

Las gramáticas de atributos son extensiones de las gramáticas incontextuales clásicas. Una *gramática incontextual* especifica las construcciones sintácticas de un lenguaje formal en términos de secuencias de otras construcciones sintácticas constituyentes. Para ello utiliza *reglas sintácticas* o *producciones*. Por ejemplo, la producción `Expr ::= Expr + Term` en la Figura 2.20 indica que una expresión aritmética (`Expr`) puede formarse concatenando una

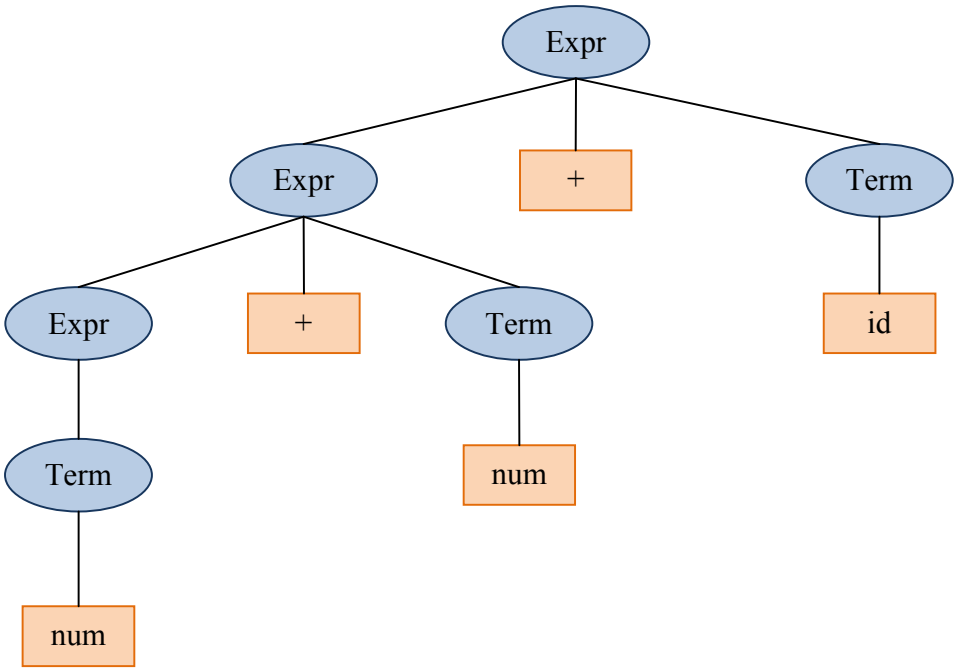
expresión aritmética, con un símbolo +, con un término (`Term`). Dado que una misma construcción sintáctica puede estructurarse de diferentes formas, es posible especificar varias producciones para una misma estructura (por ejemplo, `Expr::= Term` es otra posible regla sintáctica para la construcción `Expr` en la Figura 2.20). La construcciones primitivas (aquellas cuya estructura no se explica mediante reglas sintácticas) se denominan *terminales* (por ejemplo, `+` o `num` en la Figura 2.20), mientras que las construcciones no primitivas se denominan *no terminales* (por ejemplo, `Expr`). Las gramáticas incontextuales tienen un no terminal distinguido, denominado *símbolo inicial* o *axioma*, que se identifica con la estructura sintáctica de más alto nivel (en el caso de la gramática incontextual subyacente a la Figura 2.20, `Expr`).

Figura 2.20. Gramática de atributos para la evaluación de expresiones aritméticas sencillas

```
Expr ::= Expr + Term
   Expr0.val = Expr1.val + Term.val
   Expr1.envh = Expr0.envh
   Term.envh = Expr0.envh
Expr ::= Term
   Term.envh = Expr.envh
   Expr.val = Term.val
Term ::= num
   Term.val = str2int(num.lex)
Term ::= id
   Term.val = findVar(id.lex,Term.envh)
```

Toda gramática incontextual define un lenguaje formal: el conjunto de sentencias (secuencias de terminales) derivables de su axioma por reescrituras sucesivas de símbolos no terminales utilizando las producciones que los definen. Cada una de estas sentencias tiene asociado al menos un *árbol de análisis sintáctico*. Este es un árbol ordenado, cuya raíz es el axioma. Los nodos interiores están etiquetados por no terminales, mientras que los nodos hoja por terminales. Así mismo, dichos nodos hoja, leídos de izquierda a derecha, se corresponden con la sentencia. Por su parte, los hijos de cada nodo interior etiquetados por un no terminal, leídos de izquierda a derecha, deben corresponderse con la parte derecha de una de las reglas de dicho no terminal. La Figura 2.21 muestra un ejemplo de árbol de análisis sintáctico para la sentencia `num+num+id`, de acuerdo con la gramática incontextual subyacente a la Figura 2.20.

Figura 2.21. Árbol sintáctico de la frase "num+num+id" de acuerdo con la gramática descrita en la Figura 2.20.



Una gramática de atributos añade un conjunto de *atributos semánticos* a los símbolos de una gramática incontextual, y un conjunto de *ecuaciones semánticas* a las reglas sintácticas de dicha gramática. La asociación de atributos semánticos con los símbolos de la gramática permite asociar correspondientes *instancias de atributos semánticos* a los nodos de los árboles de análisis sintáctico. Así mismo, las ecuaciones semánticas permitirán explicar cómo se computan los valores de las instancias de las *ocurrencias* de los atributos semánticos en los diferentes símbolos de cada producción. En adelante, utilizaremos el término *atributo* para referirnos tanto a las instancias de los atributos en los nodos de los árboles, a las ocurrencias de los atributos en las producciones, y a los atributos en sí, siempre y cuando quede claro el concepto al que se hace referencia a partir del contexto. Los atributos semánticos, a su vez, se subdividen en dos categorías:

- Atributos *sintetizados*: Permiten asociar carga semántica, o significado, a los símbolos de la gramática. Por ejemplo, en la gramática de la Figura 2.20, `val` es un atributo sintetizado de las expresiones (`Expr`). Su propósito es almacenar el resultado de evaluar dichas expresiones.

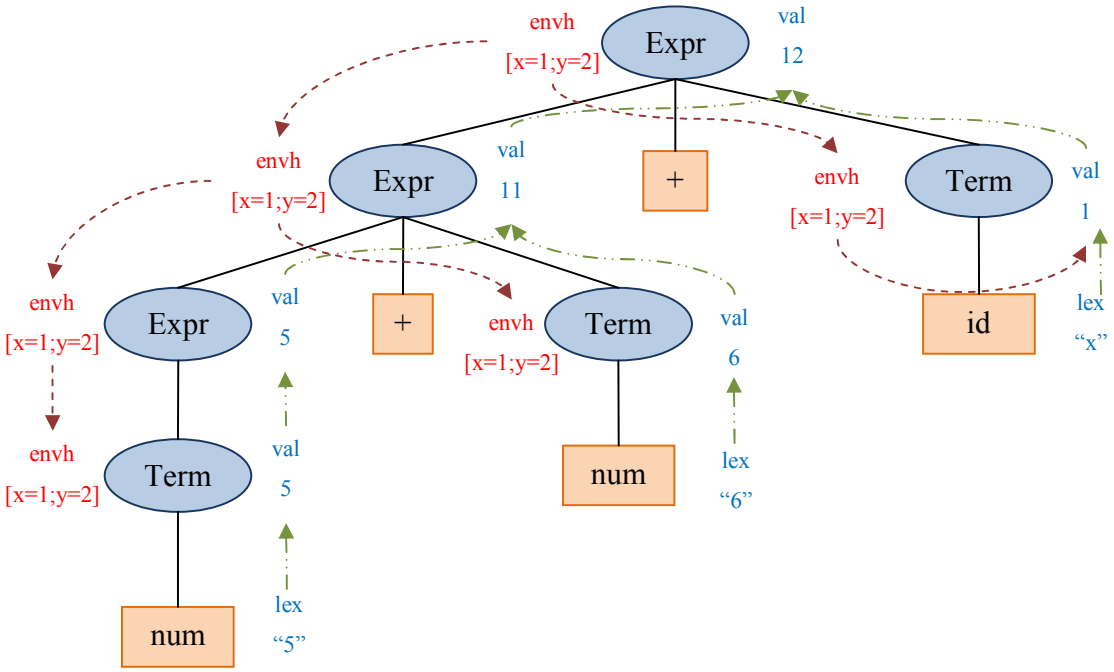
- Atributos *heredados*: Permiten asociar información de contexto a dichos símbolos. Por ejemplo, en la gramática de la Figura 2.20, `envh` es un atributo heredado que se necesita en las expresiones para especificar el valor que tienen las variables que aparecen en las mismas (dicha especificación puede llevarse a cabo, por ejemplo, mediante un conjunto de pares *variable – valor*).

De esta forma, el modelo de procesamiento que subyace a las gramáticas de atributos se basa en el cómputo de los valores de atributos en los árboles de análisis sintáctico a partir de los valores de otros atributos en dichos árboles. Para especificar las reglas que rigen dicho cómputo se utilizan *ecuaciones semánticas* en las producciones. Más concretamente, en cada producción habrá:

- Una ecuación por cada atributo sintetizado de la parte izquierda de dicha regla. La parte izquierda de dicha ecuación será una referencia al correspondiente atributo sintetizado, mientras que la parte derecha será una expresión funcional que puede referir: (i) bien atributos heredados de la parte izquierda de la regla, (ii) bien atributos sintetizados de los símbolos de la parte derecha. La definición de las *funciones semánticas* que intervienen en dicha expresión no es estrictamente parte de la gramática de atributos (dichas funciones se suponen definidas externamente). Por ejemplo, la ecuación `Term.val = findVar(id.lex,Term.envh)` en la regla `Term ::= id` de la Figura 2.20 indica que "para calcular el valor de un término que es un identificador, debe aplicarse la función `findVar` al atributo `lex` del identificador –dicho atributo contiene el nombre de la variable en sí, como se verá más adelante– y al atributo heredado `envh` del propio término". Dicha función semántica (`findVar`) podrá buscar el valor de la variable que recibe como primer argumento en la tabla de variables que recibe como segundo argumento (aunque, tal y como ya se ha indicado, dicho comportamiento deberá venir dado de manera externa a la gramática de atributos).
- Una ecuación por cada atributo heredado de los símbolos de la parte derecha. La estructura de dicha ecuación es análoga a la ya descrita, con las mismas restricciones en cuanto al uso de atributos en la expresión funcional que describe el cálculo a llevar a cabo.

Nótese que el cálculo de los atributos heredados del axioma y de los sintetizados de los terminales (*atributos léxicos*) deberá venir determinado externamente. En particular, en un procesador de lenguaje usual, los atributos léxicos serán proporcionados por el analizador léxico cuando éste convierta las secuencias de caracteres a secuencias de componentes léxicos (*tokens*) [3]. En concreto, se asumirá que aquellos no terminales que, a nivel léxico, puedan estar asociados con más de una cadena (por ejemplo, `id` o `num` en la gramática de la Figura 2.20), tendrán siempre asociado un atributo léxico `lex` que contiene la secuencia de caracteres concreta. Así por ejemplo, cuando el analizador léxico procesa una entrada como `5+6+x`, podrá generar una secuencia de *tokens* de la forma `[t:num,lex:"5"][t:+][t:num,lex:"6"][t:+][t:id,lex:"x"]`, cadena que servirá como base para construir el árbol de análisis sintáctico, y también para inicializar los valores del atributo léxico `lex` en las correspondientes hojas.

Figura 2.22. Árbol de análisis decorado y grafo de dependencias entre atributos derivados del procesamiento de "5+6+x" de acuerdo con la gramática de la Figura 2.20.



Una vez fijados los valores de los atributos heredados de la raíz, y de los atributos léxicos en las hojas[4], el cómputo del resto de los atributos se lleva a cabo aplicando las ecuaciones semánticas en un orden adecuado. La única restricción para la aplicación de dichas ecuaciones es que, para calcular el valor de un atributo, deben haberse calculado previamente los valores de los atributos requeridos por la parte derecha de la correspondiente ecuación. De esta forma, el orden de este proceso (denominado *evaluación semántica*) no es único, y el cómputo puede proceder dirigido por las *dependencias* entre los atributos del árbol. Una forma habitual de representar dichas dependencias es en forma de *grafo de dependencias*: los nodos de dicho grafo son los atributos del árbol, mientras que un arco que vaya de n_0 a n_1 indicará que el valor de n_0 es necesario para calcular el valor de n_1. La Figura 2.22 muestra el grafo de dependencias para los atributos del árbol mostrado en la Figura 2.21 (asumiendo que la sentencia original procesada fue 5+6+x), junto con los valores finalmente asignados a los mismos una vez finalizado el proceso de evaluación semántica.

Una posible estrategia de evaluación semántica es: (i) encontrar un orden topológico de los nodos del grafo de dependencias, (ii) aplicar las ecuaciones apropiadas de acuerdo con dicho orden [3]. No obstante, dicha estrategia es una de las muchas posibles, y, de hecho, una de las principales características del formalismo es no imponer a priori ningún orden de evaluación sobre los atributos. El objetivo de *Evaluators* será, precisamente, incidir sobre los conceptos básicos subyacentes a las gramáticas de atributos y a los mecanismos de evaluación semántica dirigidos por dependencias mediante la emulación del proceso de evaluación semántica.

2.6 A Modo de Conclusión

A lo largo de este capítulo se han revisado distintos conceptos que, en menor o mayor medida, están relacionados con el campo de trabajo de este proyecto de fin de máster:

- En primer lugar se ha presentado una visión general de las distintas metodologías y técnicas que se utilizan más habitualmente a la hora de impartir las materias relacionadas con la creación de Procesadores de Lenguaje. Esta revisión ha

[4] Desde un punto de vista matemático, estos valores pueden interpretarse como *condiciones de contorno* para la resolución del sistema de ecuaciones inducido sobre el árbol por la gramática de atributos.

permitido estudiar las principales ventajas e inconvenientes de estos métodos de enseñanza y situar el contexto en el que se empleará *Evaluators*.

- Posteriormente, se han analizado distintas herramientas software educativas centradas en la enseñanza de distintos conceptos y materias pertenecientes a la informática. Con este estudio se ha perseguido conocer las principales características que definen a las herramientas software educativas destinadas a la enseñanza de la informática. Este estudio ha evidenciado la importancia de diseñar mecanismos de visualización adecuados que faciliten el aprendizaje de los distintos conceptos involucrados en cada dominio.
- Después, se han estudiado distintas herramientas software educativas centradas en la enseñanza de las materias impartidas durante un curso de construcción de compiladores. La principal, y más importante, conclusión extraída de ese análisis es que la mayoría de las herramientas se centran en mejorar la experiencia educativa de los alumnos en las últimas fases del curso, así como en los aspectos relativos a la implementación de los procesadores, incidiendo escasamente en los aspectos de especificación y en la enseñanza de los conceptos básicos de la materia.
- Seguidamente se han revisado distintos videojuegos educativos aplicados a distintas materias relacionadas con la informática. En este análisis se ha hecho patente la importancia de encontrar metáforas adecuadas que permitan mapear los conceptos y procesos a enseñar en la estructura y la dinámica de los correspondientes videojuegos.
- Para finalizar, se ha revisado con cierto detalle el formalismo de las gramáticas de atributos, ya que *Evalutors* se centrará en la enseñanza de los conceptos básicos subyacentes a dicho formalismo.

De esta forma, a lo largo de este capítulo se ha proporcionado el contexto necesario para entender a *Evaluators* como una herramienta software educativa, y, más concretamente, como un videojuego educativo Dicha herramienta estará centrada en la enseñanza de los conceptos básicos del formalismo de las gramáticas de atributos. El próximo capítulo describe esta herramienta.

Capítulo 3. Propuesta

3.1 Introducción

En las principales recomendaciones para el currículo académico de la Ingeniería Informática se apunta a la implementación de lenguajes de computadores como un aspecto clave a incluir en la carrera [1]. Por ese motivo, *Procesadores de Lenguaje* es una asignatura troncal en la actual titulación de la Ingeniería Informática [13], y aparece claramente reflejada en las recomendaciones para el módulo de Computación en los incipientes grados de Ingeniería Informática [12], estando presente en la mayoría de planes para dichos grados en las diferentes universidades españolas.

En la Universidad Complutense de Madrid, en la impartición de la asignatura de *Procesadores de Lenguajes* se aboga por un proceso de construcción de compiladores en el que esté claramente separado el diseño y especificación del lenguaje de la subsecuente implementación. Para ello, se adopta la *traducción dirigida por sintaxis* como paradigma central para construir los procesadores de lenguaje y el formalismo de las gramáticas de atributos (véase sección 2.5) como modelo básico para la especificación de estos.

Los docentes de la asignatura en la UCM, con el fin de reforzar los conceptos básicos del formalismo, comienzan empleando baterías de ejercicios relacionados con los niveles más bajos de la taxonomía de Bloom (en particular, con el nivel de *comprensión*) [11]. Un ejercicio típico en estos niveles iniciales del aprendizaje consiste en[5]: (i) descripciones informales de problemas de procesamiento de lenguaje, (ii) descripciones formales mediante gramáticas de atributos, y (iii) sentencias del lenguaje para ser procesadas. Los estudiantes deben encontrar, para cada problema de procesamiento y sentencia, el árbol sintáctico producto del procesamiento de la sentencia y explicar cómo evaluar los atributos semánticos de cada nodo.

A pesar de que esta aproximación tiene un resultado positivo en el aprendizaje de los alumnos, que asimilan los conceptos básicos del formalismo, este efecto no es tan bueno como se podría desear. En particular, se ha detectado un alarmante predominio de aprendizaje memorístico. De hecho, la mayoría de los estudiantes se limita a extrapolar las plantillas de solución dadas en clase al resto de ejercicios. Es por esto que los docentes encargados de

[5] Tipo de ejercicio sugerido por los Profesores Alfredo Fernández-Valmayor y José Luis Sierra.

impartir esta asignatura son conscientes de la necesidad de desarrollar métodos de aprendizaje más efectivos para enseñar el formalismo de las gramáticas de atributos.

En este trabajo de investigación, alentados por la experiencia previa promocionando el aprendizaje activo en disciplinas relacionadas con el procesamiento de lenguaje mediante herramientas especializadas [63] y basándonos en otras herramientas y simuladores aplicados a la enseñanza de la informática (véase la revisión realizada en el Capítulo 2), proponemos un entorno de simulación donde los estudiantes pueden resolver ejercicios y obtener comentarios sobre sus éxitos y errores durante la resolución. Teniendo en cuenta la componente altamente motivadora de los videojuegos aplicados al aprendizaje, se ha decidido usar un sistema de producción de videojuegos en donde la misión de los estudiantes esté directamente relacionada con la solución de ejercicios de procesamiento de lenguajes. Dicho sistema se denomina, tal y como ya se ha indicado, *Evaluators*.

Las siguientes secciones están organizadas de la siguiente manera: en primer lugar se introducirá un modelo genérico para construir herramientas de enseñanza de informática basadas en videojuegos generados a partir de ejercicios. Después se explicará en detalle el sistema *Evaluators*, y cómo dicho sistema se ha desarrollado siguiendo el modelo de proceso genérico introducido.

3.2 Un Modelo de Proceso para la Producción de Videojuegos Educativos Dirigidos por Ejercicios

En esta sección proponemos un modelo de proceso para la producción de videojuegos educativos guiados por los ejercicios propuestos por los docentes en clase. Para ello suponemos que:

- Los docentes adoptan una estrategia de aprendizaje basada en problemas, y que proporcionan a sus alumnos ejercicios de un determinado tipo que trabajan diferentes aspectos de la materia a enseñar.
- El proceso de resolución de dichos ejercicios puede replantearse de manera significativa como un videojuego.

El modelo de proceso se expone de forma deliberadamente general, de manera que sea aplicable, no únicamente a la enseñanza / aprendizaje de los procesadores de lenguaje, sino también a otros campos. Dicho modelo se inspira, parcialmente, en la experiencia ganada con el

desarrollo de *Evaluators*, así como la observada en otros enfoques similares (por ejemplo, [27] [34]).

Siguiendo los enfoques de estructuración de modelos de producción de herramientas y contenidos en e-Learning descritos en [64][65], el modelo se describirá en términos de tres perspectivas distintas: productos y actividades, secuenciación de actividades y los actores del modelo con sus distintos roles. El modelo en sí se describe también en [56].

3.2.1 Productos y Actividades

Los productos y actividades involucrados en nuestro modelo de producción dirigido por ejercicios se esbozan en la Figura 3.1. De esta forma:

- El objetivo de la actividad de *concepción* es la de crear los modelos del dominio que guiarán el resto del proceso. Por un lado, es necesario crear un modelo de ejercicios que permita a los alumnos reforzar, y profundizar en, los conceptos aprendidos en clase: el *modelo del ejercicio*. Este modelo se caracterizará por la descripción del ejercicio y su solución. Por otro lado, es necesario modelar los videojuegos educativos basándose en los procesos mentales requeridos para resolver esa clase de ejercicios. El *modelo del videojuego* contiene esa información. Este modelo estará en gran parte basado en una *metáfora* adecuada que permita transformar los ejercicios, junto con sus soluciones, en videojuegos (véase, por ejemplo, [28] para un análisis más profundo del papel jugado por las metáforas en la creación de videojuegos educativos).
- La actividad de *creación de herramientas* está orientada a proporcionar las aplicaciones necesarias para implementar satisfactoriamente el sistema de aprendizaje basado en videojuegos: las *herramientas de autoría*, el *generador de videojuegos* y las *herramientas de análisis*. Esta actividad usa el modelo de ejercicios para producir las herramientas de autoría, mientras que los dos modelos, de ejercicios y del juego, son empleados para desarrollar el generador de videojuegos y las herramientas de análisis.
- Los docentes usan las herramientas de autoría en la actividad de *creación de ejercicios* para crear colecciones de ejercicios que podrán ser usados para crear videojuegos educativos. Por otro lado, el generador de videojuegos será capaz de

generar automáticamente juegos a partir de los ejercicios (junto con sus soluciones); los cimientos de este proceso de generación se encuentran en la metáfora del modelo del videojuego. La generación por si misma se lleva a cabo en la actividad de *generación del videojuego*. Finalmente, la herramienta de análisis permite a los docentes estudiar el comportamiento de los alumnos dentro del videojuego durante la actividad *uso del videojuego*. De hecho, como resultado de esta actividad, los videojuegos registran el comportamiento de los estudiantes en un archivo de *log*, *logs* que pueden ser analizados con las herramientas de análisis durante la actividad de *evaluación de los estudiantes* para evaluar el progreso de dichos estudiantes.

Cabe destacar también aquí cómo es posible superponer una valoración formativa, orientada a mejorar el sistema de aprendizaje de forma paralela a las diferentes actividades del modelo de proceso. De hecho, los docentes pueden valorar diferentes aspectos de las herramientas de autoría durante la actividad de creación de ejercicios (por ejemplo, ¿la herramienta es fácil de usar?, ¿la herramienta proporciona suficiente expresividad para crear los ejercicios que necesito?, etc.). Por otro lado, los estudiantes pueden evaluar diferentes aspectos del videojuego durante la actividad de uso del videojuego (por ejemplo, usabilidad, grado de entretenimiento, utilidad pedagógica subjetiva, etc.). Finalmente, los docentes pueden evaluar la calidad de la herramienta de análisis y la eficacia pedagógica de todo el sistema de aprendizaje durante la actividad de *evaluación de los alumnos*.

Finalmente, y desde un punto de vista del desarrollo, es interesante notar cómo, en este modelo de proceso, juegan un papel esencial los enfoques de desarrollo de software basados en programación generativa [15], y los enfoques de desarrollo de software dirigidos por lenguajes [23][37], principalmente en lo que se refiere a la vertiente moderna de los mismos basada en los principios de la Ingeniería del Software dirigida por modelos [68]. Efectivamente:

- El modelo del ejercicio puede concebirse como una caracterización explícita de la sintaxis abstracta para un lenguaje específico de dominio que permite a los docentes describir ejercicios relativos a un determinado campo de enseñanza / aprendizaje. Por su parte, la herramienta de autoría puede concebirse como un editor para dicho lenguaje, que implementa una sintaxis concreta adecuada para el mismo.

- El modelo del videojuego, por su parte, puede concebirse como un lenguaje específico para la descripción de un determinado tipo de videojuego. Por su parte, el generador del juego se basa en una transformación o cadena de transformaciones adecuada entre el modelo del ejercicio y el modelo del videojuego.
- Por último, la semántica operacional del modelo del videojuego proporcionará el fundamento básico para la ejecución de los juegos generados.

De esa forma, el modelo de proceso presentado hereda los principios de desarrollo de sistemas educativos dirigido por lenguajes propugnado en [44][46][66][67].

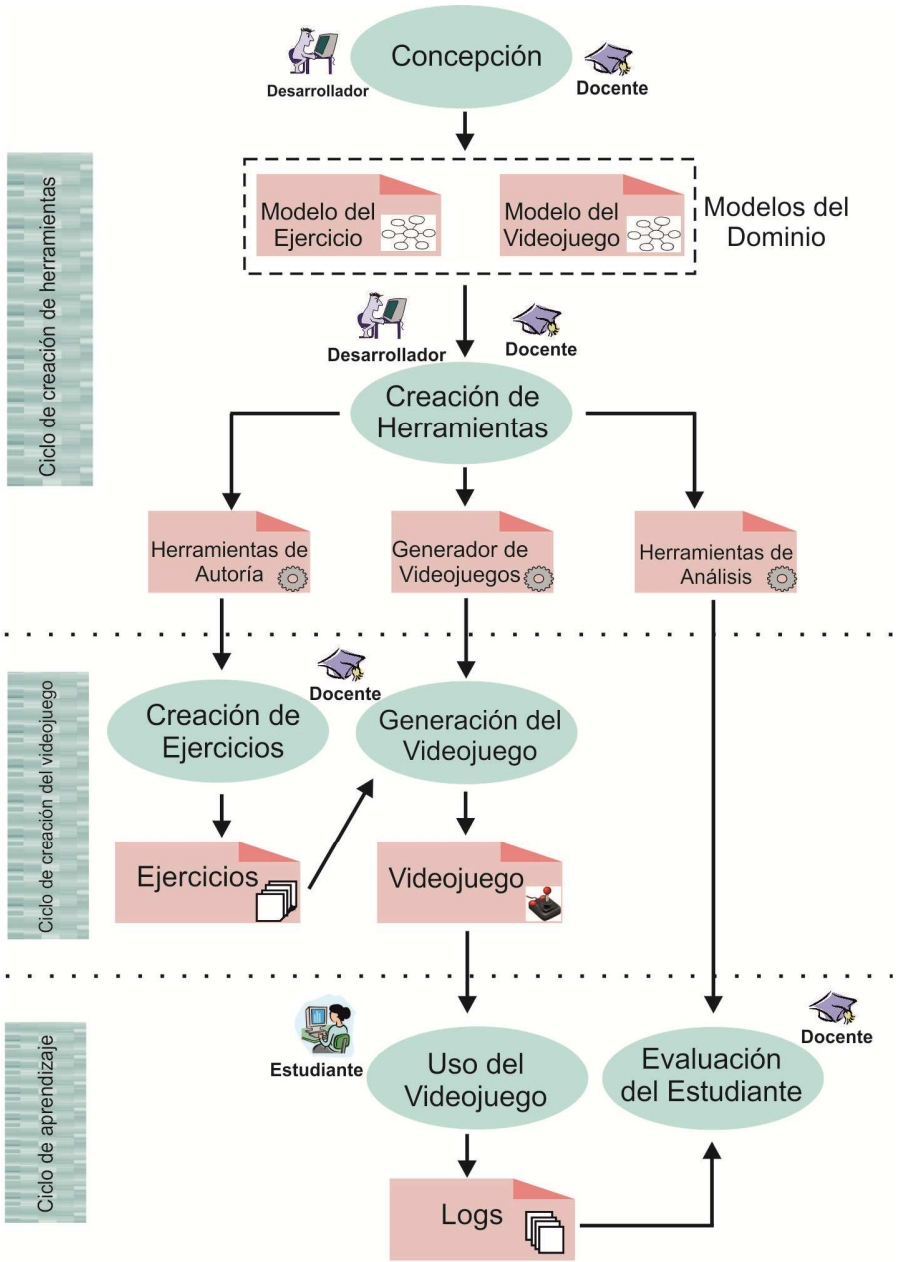
3.2.2 Secuenciación de las Actividades

Las actividades introducidas en la sección anterior están secuenciadas tal y como se muestra en la Figura 3.1. De acuerdo con este esquema, el modelo de proceso introduce tres ciclos diferentes:

- *El ciclo de creación de herramientas.* Este ciclo se caracteriza por la continua iteración de las actividades de concepción y creación de herramientas, con el fin de producir los diferentes instrumentos que apoyen la estrategia de aprendizaje basada en videojuegos: las herramientas de autoría, el generador de videojuegos y las herramientas de análisis. Cuando este ciclo finaliza se obtienen modelos de dominio y herramientas de apoyo estables.
- *El ciclo de creación de videojuegos.* Este ciclo aparece con la iteración de las actividades de creación de ejercicios y creación de videojuegos con el fin de crear un repositorio de ejercicios adecuado y su correspondiente videojuego.
- *El ciclo de aprendizaje.* Este ciclo se identifica por el uso del videojuego educativo. Durante este ciclo, los estudiantes usan el videojuego, y los docentes valoran su progreso.

Además, el modelo de proceso contempla la posibilidad de moverse desde los ciclos finales del proceso a los ciclos iniciales. Esto puede deberse, por ejemplo, a la identificación, durante la evaluación formativa del sistema de enseñanza / aprendizaje de alguna oportunidad de mejorar los diferentes instrumentos implicados en la estrategia de aprendizaje. De hecho:

Figura 3.1. Esquema de los productos, actividades y actores del modelo de proceso.



- Durante el ciclo de aprendizaje es posible realizar diferentes encuestas a los estudiantes para evaluar los videojuegos. Si estas encuestas revelan algún aspecto que deba ser mejorado en los videojuegos, se reactivará el ciclo de creación de videojuegos. Esta reactivación puede ser también consecuencia de la identificación de algunas deficiencias en el progreso de los estudiantes durante la

actividad de evaluación de estudiantes. Además, durante este ciclo podrían observarse también algunas deficiencias en el videojuego o en las herramientas de análisis, que deberían ser arregladas. Esto llevaría a la reactivación del ciclo de creación de herramientas.

- Durante el ciclo de creación de videojuegos es posible descubrir algunas carencias en la expresividad de las herramientas de autoría, así como errores técnicos en estas herramientas o en el generador de videojuegos. Todas estas circunstancias pueden llevar a la reactivación del bucle de creación de herramientas.

3.2.3 Actores y Roles

Como se indica en la Figura 3.1, el modelo de proceso contempla tres actores diferentes: docentes, desarrolladores y estudiantes. Mientras que los docentes se enfrentan con los aspectos pedagógicos, los desarrolladores se preocupan de los aspectos más tecnológicos. Además, los docentes guían a los desarrolladores en el diseño de las herramientas de autoría y análisis, ya que dichos docentes serán los que posteriormente las usarán. Por otro lado, los estudiantes usan el videojuego y sus consejos y opiniones son esenciales para detectar deficiencias que deben ser subsanadas para obtener un sistema de aprendizaje competente y útil. De esta forma:

- La mayor carga de trabajo de los desarrolladores se localiza durante las actividades de concepción y creación de las herramientas, ya que son responsables de formalizar los modelos del dominio, así como desarrollar las herramientas de apoyo (herramientas de autoría y análisis, y el generador del videojuego).
- Por otro lado, los docentes juegan un papel relevante durante la mayoría de las actividades del modelo de proceso. Durante la concepción y la creación de las herramientas contribuyen con su conocimiento del dominio. Por lo tanto, juegan un papel esencial determinando la estructura de los ejercicios y las soluciones, así como en la creación de una metáfora de juego adecuada. Durante las actividades de creación de ejercicios y creación de videojuegos se encargan de la producción de ejercicios, y, por lo tanto, la generación del videojuego. Finalmente, durante la evaluación de los estudiantes, evalúan el progreso de los mismos.
- Para terminar, los estudiantes participan principalmente jugando con los videojuegos (o, lo que es lo mismo, resolviendo los ejercicios). Además de

resolver los ejercicios propuestos, los estudiantes juegan un papel relevante en la evaluación de los videojuegos generados, contribuyendo así a la mejora de la calidad de dichos juegos.

3.3 Concepción de *Evaluators*

Evaluators es un sistema de aprendizaje basado en videojuegos dirigido por ejercicios que está orientado a enseñar conceptos fundamentales de la implementación de lenguajes. El objetivo de *Evaluators* es ayudar a entender conceptos básicos del formalismo de las gramáticas de atributos. Este videojuego ha sido desarrollado siguiendo el modelo de proceso descrito en la sección anterior. Por tanto, la primera fase en su desarrollo es la de su concepción.

Tal y como se ha indicado en el capítulo anterior, el modelo computacional de las gramáticas de atributos se basa en construir un árbol sintáctico para cada sentencia del lenguaje, en asociar atributos semánticos a los nodos de dicho árbol, y en llevar a cabo un proceso de evaluación semántica para determinar, en base a las ecuaciones semánticas, los valores de dichos atributos. Así mismo, es importante recalcar que las gramáticas de atributos no imponen un orden a la hora de computar el valor de cada uno de los atributos de un nodo: la única restricción que debe ser satisfecha es que, antes de computar el valor de un atributo, deben haberse computado los valores de todos los atributos de los que éste depende. Por lo tanto, es muy importante que los alumnos interioricen este modelo de cómputo, como paso necesario para la posterior elaboración de sus propias especificaciones basadas en gramáticas de atributos. Para este propósito, tal y como ya se ha anticipado, los profesores de Procesadores de Lenguaje en la UCM sugieren utilizar los siguientes tipos de ejercicios:

- La descripción del ejercicio consta de problemas de procesamiento de lenguajes, formalizaciones de dichos problemas basadas en gramáticas de atributos, y sentencias a procesar.
- La solución al ejercicio consiste en determinar, para cada problema y sentencia, un posible orden en el que los atributos del árbol sintáctico producto de procesar la sentencia puedan ser evaluados, así como el valor final de estos atributos.

De esta forma, en *Evaluators* los videojuegos generados están basados en este tipo de ejercicios. De hecho, el modelo de ejercicios de *Evaluators* se ajusta a la especificación descrita anteriormente. Por tanto, las soluciones de los mismos incluirán el árbol sintáctico decorado con

las dependencias entre los atributos y sus valores. Por su parte, el modelo del videojuego en *Evaluators* concibe juegos en los que el alumno debe llevar a cabo el proceso de evaluación semántica. A continuación se describen ambos modelos.

3.3.1 Modelo de ejercicios

La Figura 3.2 muestra una formalización inicial, en forma de diagrama de clases UML, de este modelo de ejercicio. En este modelo cabe resaltar los siguientes aspectos:

- Cada ejercicio consta de una secuencia de supuestos de procesamiento de lenguaje (clase `Supuesto`). Cada supuesto, a su vez, almacena en uno de sus campos la descripción informal y formal del problema de procesamiento, y refiere a una representación estructurada de la solución.
- Dicha representación de la solución viene dada como un árbol de análisis sintáctico atribuido, que incluye, además, el grafo de dependencias y los valores de los atributos. El modelo del árbol sintáctico decorado viene representado por las clases `Nodo`, `NodoT`, `NodoNT` y `Atributo`, donde cada atributo contendrá un campo orientado a almacenar el valor que le corresponda, y dependerá explícitamente de los atributos que se precisan para calcular dicho valor (dicha relación de dependencia se representa también explícitamente en el modelo). Así mismo, los nodos no terminales (`NodoNT`) tendrán también un campo que permite almacenar la producción y las ecuaciones semánticas asociadas al contexto que estos representan.

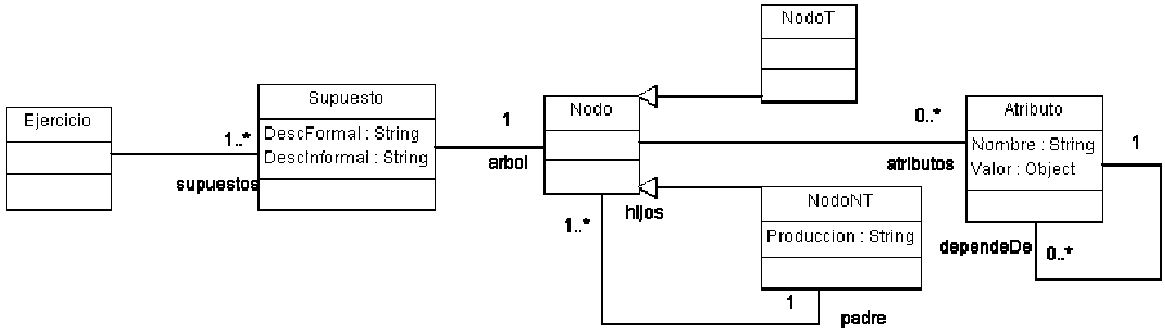
3.3.2 Modelo de videojuegos

El modelo del juego, esbozado en la Figura 3.3 como otro diagrama de clases UML[6], está basado en una metáfora que adopta las siguientes convenciones:

- Los arboles sintácticos se visualizan como mundos laberínticos donde las habitaciones representan los nodos del árbol de análisis sintáctico y los arcos de dicho árbol se representan como pasillos.

[6] En este esbozo se incluyen únicamente los elementos más relevantes para la concepción, obviando otros que no son estrictamente esenciales para la misma (como, por ejemplo, los oráculos encargados de proporcionar información contextual durante el proceso de juego).

Figura 3.2. Modelo del ejercicio esbozado como un diagrama de clases UML.



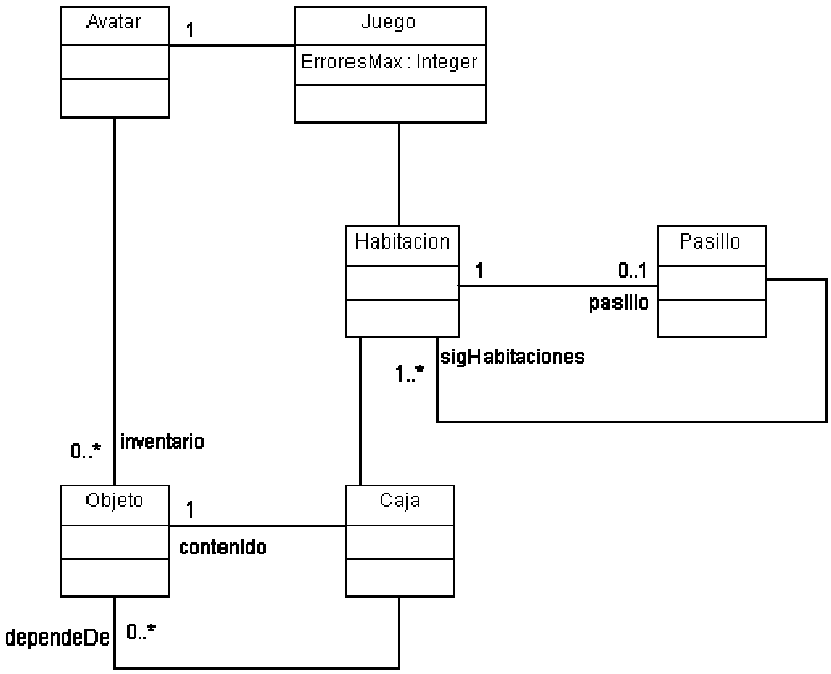
- Dentro de cada habitación se encuentra una mesa con cajas sobre ella. Estas cajas representan instancias de atributos semánticos y contienen objetos, que son los valores de dichos atributos.
- Finalmente, cada juego incluye un avatar que el estudiante puede manejar por el laberinto. Usando este avatar el estudiante puede transportar objetos de caja a caja para poder reproducir el proceso de evaluación de atributos.

El principal objetivo en el videojuego es mover los contenidos de las cajas apropiadas a través del laberinto y combinarlas para activar todas las cajas. La metáfora educacional mapea el problema especificado por el tutor en un videojuego donde la acción de activar una caja es una forma metafórica de expresar que se ha computado el valor de un atributo.

3.4 Creación de las Herramientas de *Evaluators*

En esta sección, y siguiendo el modelo de proceso descrito anteriormente, se describen las herramientas software asociadas al sistema de aprendizaje *Evaluators*. Estas herramientas se derivan, en gran medida, de los modelos del dominio descritos previamente. De esta forma, primero se describe la herramienta de autoría creada, después el generador de videojuegos y por último la herramienta de análisis.

Figura 3.3. Modelo del juego esbozado como un diagrama de clases UML.



3.4.1 Herramienta de Autoría

La herramienta de autoría permite a los docentes codificar los ejercicios en un formato que *Evaluators* entienda y con el que sea capaz de crear los laberintos que posteriormente tendrán que superar los alumnos.

La primera versión de dicha herramienta consistía en un editor que obligaba al docente a construir el árbol sintáctico decorado explícitamente (véase [22]; en el próximo capítulo se proporcionan también algunos detalles adicionales acerca de dicha versión). Esta tarea podía tornarse harto ardua a medida que el árbol sintáctico crecía en tamaño. Por ello, como parte de este trabajo de investigación se ha desarrollado una versión mejorada donde el docente puede especificar gramáticas de atributos en un lenguaje sencillo y muy cercano al lenguaje empleado

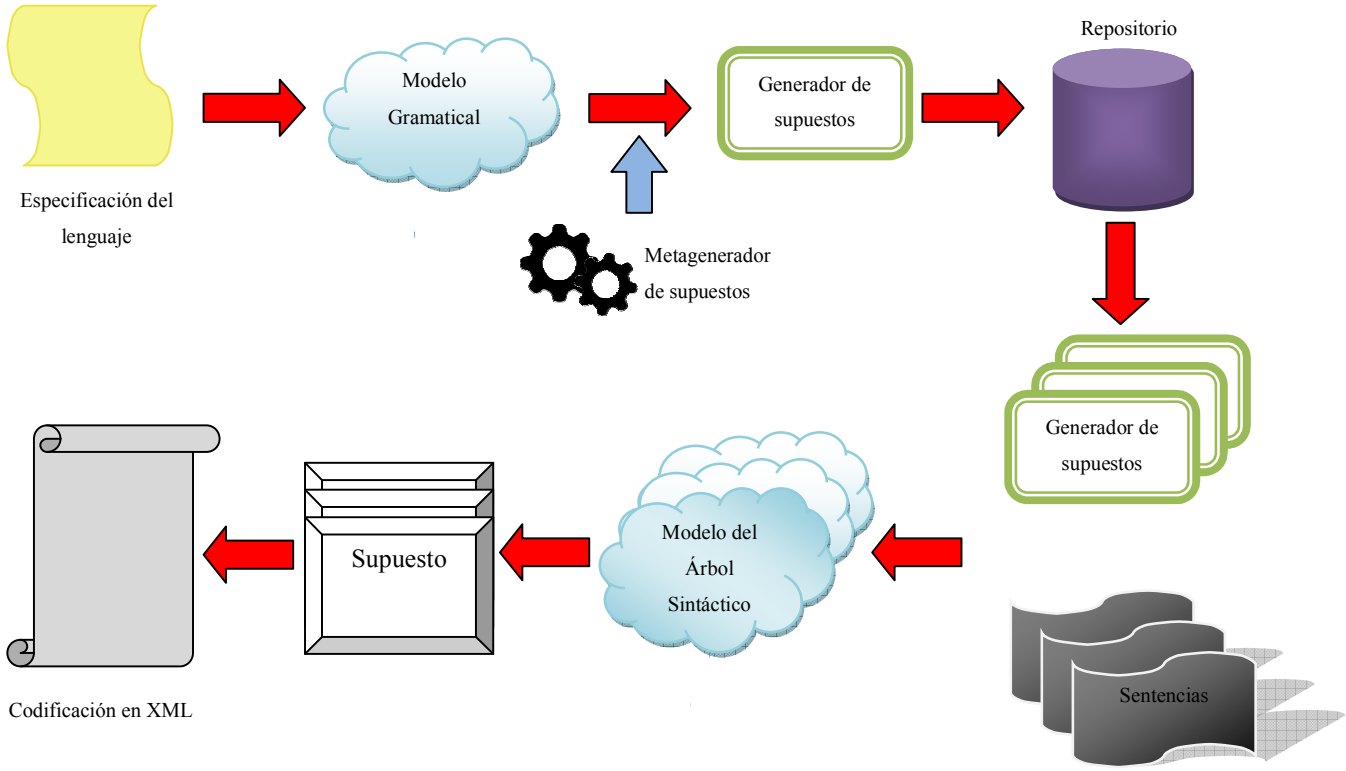
en clase para presentar este tipo de especificaciones, y después usar esas gramáticas junto con sentencias del lenguaje que especifican para componer colecciones de ejercicios.

La Figura 3.4 esboza el proceso de creación de ejercicios soportado por la actual herramienta de autoría, que es una evolución profunda del primitivo editor de ejercicios asociado a la anterior versión de *Evaluators*. De acuerdo con esta figura:

- El profesor puede describir problemas de procesamiento de lenguajes directamente en forma de gramáticas de atributos, utilizando el lenguaje de especificación al que se ha aludido anteriormente.
- Las especificaciones se utilizan para instanciar un *modelo gramatical* (que, en realidad, supone la sintaxis abstracta del lenguaje de especificación).
- Este modelo se procesa para crear un *generador de supuestos*. Para ello, la herramienta de autoría aplica un componente denominado *metagenerador de supuestos*. Dicho componente se apoya, de forma transparente, en dos herramientas de creación de compiladores, la pareja CUP y JFLEX [7]. Efectivamente, la herramienta genera código en JFLEX y en CUP a partir del modelo gramatical, código que, una vez procesado por las citadas herramientas, da lugar al citado generador de supuestos. Este generador es, en realidad, un procesador que acepta sentencias del lenguaje definido por la gramática de atributos, y que, como resultado de procesar las mismas, instancia adecuadamente los correspondientes supuestos de acuerdo con el modelo del ejercicio, generando los árboles atribuidos, con todos sus valores evaluados y con las correspondientes dependencias y demás información representadas.
- El correspondiente generador puede incorporarse a un *repositorio de generadores de supuestos*. En dicho generador se almacenará, por una parte, la descripción informal del enunciado correspondiente al problema de procesamiento, la gramática de atributos en sí, y el correspondiente generador de supuestos.
- Por último, el profesor puede generar ejercicios especificando los supuestos. Para cada supuesto deberá elegir únicamente el correspondiente generador de supuestos del respositorio, así como indicar la sentencia involucrada. La herramienta aplicará el generador de supuestos correspondiente a dicha sentencia para generar automáticamente el árbol atribuido. El supuesto en sí resultará de

combinar dicho árbol, la sentencia, el enunciado informal del problema de procesamiento, y la gramática de atributos. El ejercicio en sí se serializará como un archivo XML.

Figura 3.4. Flujo de trabajo del proceso de creación de un ejercicio.



A continuación se describe con más detalle los aspectos más relevantes que intervienen en dicho proceso. Se comienza describiendo el lenguaje de especificación, así como el modelo gramatical asociado. Se describen también el funcionamiento del metagenerador de supuestos, haciendo énfasis en los patrones utilizados por dicho metagenerador para codificar los generadores de supuestos mediante JFLEX y CUP. Por último, se presenta la interfaz de usuario de la herramienta, que facilita la aplicación del proceso de creación de ejercicios descrito.

3.4.1.1 El lenguaje de especificación

El lenguaje de especificación permite formalizar problemas de procesamiento de lenguajes como gramáticas de atributos. Más concretamente, cada especificación consta de cuatro secciones diferentes (véase Figura 3.5):

- Una sección donde se describen los distintos no terminales que compondrán el lenguaje, especificando los atributos sintetizados y los heredados asociados a cada no terminal.
- Una sección donde se describen los distintos terminales que compondrán el lenguaje, especificando si se trata de un numero (entero o decimal), una cadena de texto, una cadena de texto acotada por un carácter (por ejemplo, un *string* acotado por el símbolo “) o un literal definido por el usuario.
- Una sentencia en la que se define el no terminal que será el axioma de la gramática.
- Por último, las reglas gramaticales del lenguaje acompañadas de las ecuaciones semánticas asociadas a cada regla.

Para describir las ecuaciones semánticas el lenguaje cuenta con operaciones y tipos predefinidos estándar como son los enteros, los reales, los booleanos y las cadenas de caracteres (los *String*). Las operaciones a realizar son las propias de cada tipo (por ejemplo, la suma, la concatenación, la negación, etc.). Estos tipos, junto con sus operaciones asociadas, se corresponden con los tipos predefinidos de Java (Integer, String, Boolean y Float).

El conjunto de operaciones y tipos puede no resultar suficiente para lenguajes complejos o que precisen estructuras de datos específicas. Por ello se permite la utilización de funciones y tipos de datos externos que estarán definidos en una clase semántica externa. Esta clase contendrá la definición de los tipos en función de clases, tanto estándar de Java, como otras definidas por el usuario como clases anidadas dentro de la clase semántica, y la definición de las funciones semánticas como métodos dentro de esa clase.

Figura 3.5. Descripción BNF del lenguaje de especificación de gramáticas de atributos empleado en la herramienta de autoría.

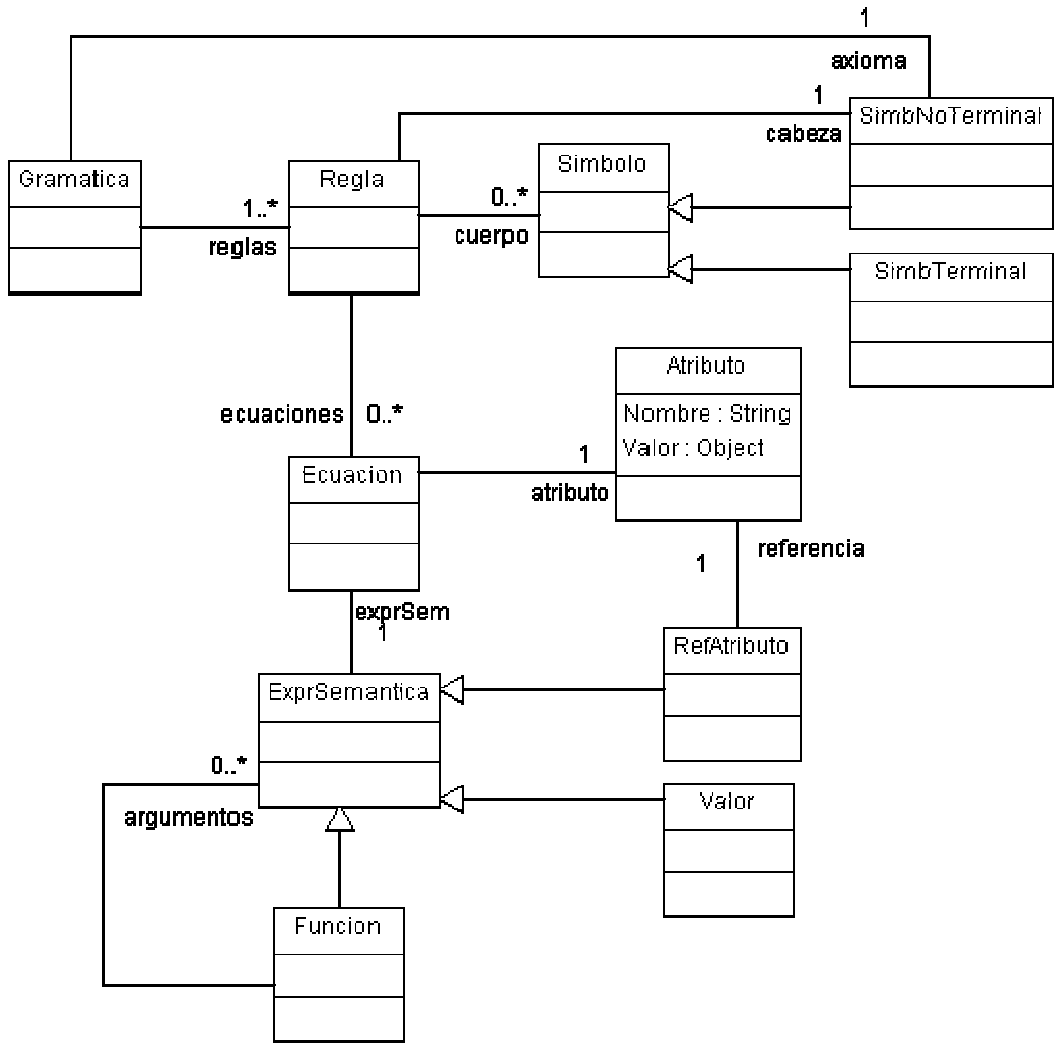
```
Grammar ::= SymbolDecs Rules
SymbolDecs ::= NTDecs AxiomDec TDecs
NTDecs ::= NTDecs NTDec
NTDecs ::= NTDec
NTDec  ::= nt ( NTName , Atts , Atts ) .
NTName ::= IDEN
AxiomDec ::= axiom ( IDEN ) .

TDecs  ::= TDecs TDec
TDecs  ::= TDec
TDec   ::= t ( TName , ExprReg ) .
TName  ::= IDEN
ExprReg ::= ID_TYPE
ExprReg ::= REAL_TYPE
ExprReg ::= INT_TYPE
ExprReg ::= QUOTED ( STRING )
ExprReg ::= LIT ( STRING )

Atts ::= [ ]
Atts ::= [ AttList ]
AttList ::= AttList , IDEN
AttList ::=  IDEN

Rules ::= Rules Rule
Rules ::= Rule

Rule ::= Lhs ::= Rhs Equations .
Lhs  ::= IDEN
Rhs  ::= <lambda>

Rhs  ::= SList
SList ::= SList IDEN
SList ::= IDEN

Equations ::=
Equations ::= -- EqList
```

```
EqList ::= EqList Equation
EqList ::= Equation

Equation ::= AttrRef = Exp

Exp ::= ExpA OpComp ExpA
Exp ::= ExpA

ExpA ::= ExpA OpAdd Term
ExpA ::= Term

Term ::= Term OpMul Fact
Term ::= Fact

Fact ::= OpUn Fact
Fact ::= Atom

Atom ::= FunApp
Atom ::= AttrRef
Atom ::= STRING
Atom ::= NUM
Atom ::= FLOAT
Atom ::= (  Exp )
FunApp ::= IDEN ( Args )
Args ::=
Args ::= ArgList
ArgList ::= ArgList , Exp
ArgList ::= Exp

AttrRef ::= IDEN . IDEN Index
Index ::= ( NUM )
Index ::=

OpComp ::=  < | > | <=| >= | == | =\=
OpAdd  ::= + | - | | | ||
OpMul  ::= & | * | /

OpUn  ::= ~ | -
```

3.4.1.2 El modelo gramatical

El modelo gramatical consta de un conjunto de clases que, convenientemente instanciadas, permiten representar en memoria las gramáticas de atributos descritas mediante el lenguaje de especificación. Desde este punto de vista, dicho modelo es, como ya se ha indicado, una caracterización explícita de la sintaxis abstracta del lenguaje de especificación.

Figura 3.6. Diagrama de clases UML que esboza el modelo gramatical.



El modelo se compone de los elementos descritos en forma de diagrama de clases UML de la Figura 3.6. Se puede apreciar que el modelo contiene los elementos necesarios para

representar las gramáticas de atributos especificadas. Más concretamente, el modelo permite representar:

- Las distintas reglas que compondrán la gramática (clase `Regla`). Cada regla se compone de un símbolo no terminal (clase `SimbNoTerminal`), que será la cabeza de la regla, y de uno o más símbolos (clase `Simbolo`), que compondrán el cuerpo de la regla (estos símbolos pueden ser tanto de la clase `SimbNoTerminal` como de la clase `SimbTerminal`). Además, cada regla tiene asociada varias ecuaciones semánticas (clase `Ecuacion`).
- Cada ecuación está compuesta por un atributo que representa la parte izquierda de la ecuación, y por una expresión semántica (`ExprSemantica`) que puede ser de tres tipos distintos: un valor, una referencia a un atributo o una función. Una función puede tener argumentos que serán a su vez otras expresiones semánticas.
- La gramática, además, especifica cuál de todos los símbolos no terminales que la componen será el axioma de la gramática.

De esta forma, el modelo incluye recursos suficientes para modelar cualquier gramática de atributos expresable mediante el lenguaje de especificación. Este modelo tiene como finalidad última asistir la creación de metageneradores de supuestos permitiendo un almacenamiento y acceso sencillo a las gramáticas de atributos definidas empleando el lenguaje de especificación anteriormente descrito.

3.4.1.3 El metagenerador de supuestos

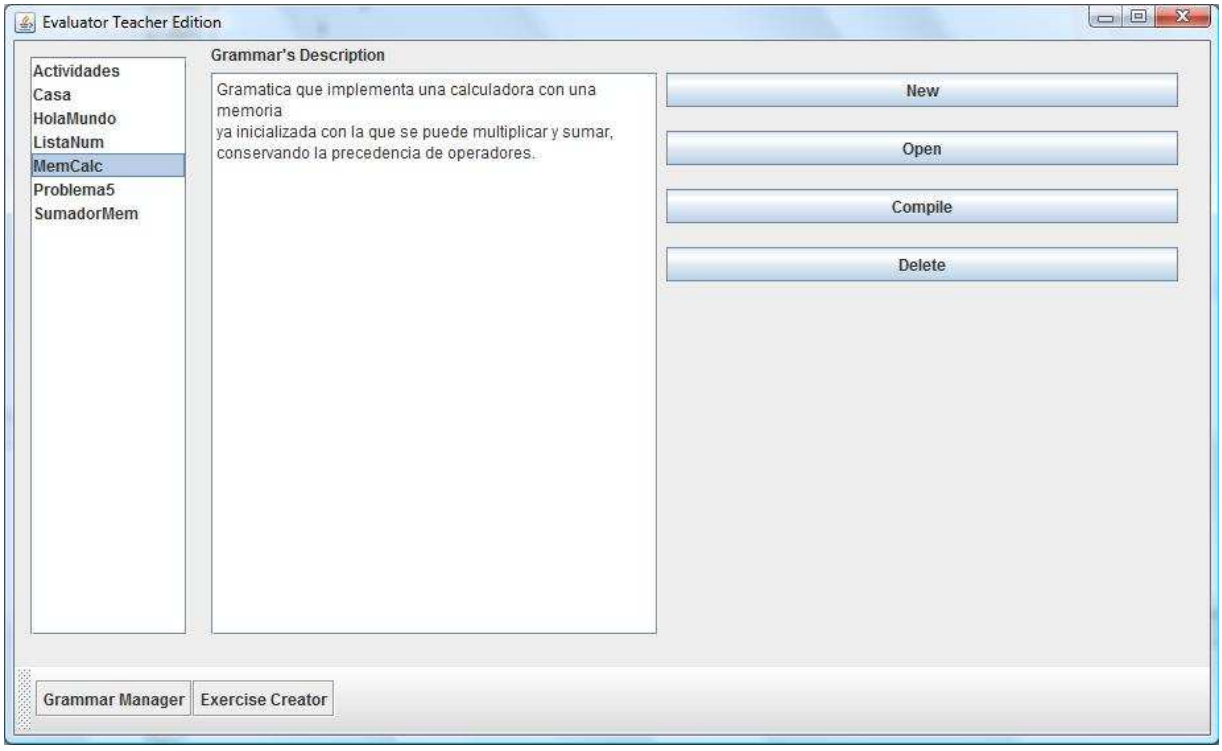
El metagenerador de supuestos es el encargado de generar los generadores de supuestos en función de la gramática de atributos especificada. De esta forma, el componente se encarga de procesar la gramática, modelada usando el modelo gramatical, y crear un generador de supuestos acorde con el lenguaje especificado. Este generador de supuestos estará basado en la pareja JFLEX/CUP. El metagenerador se encargará de especificar el generador en función de estas herramientas, de manera que dicho generador sea capaz de crear el árbol sintáctico decorado a partir de las sentencias. Para ello, el generador será codificado de tal forma que, a medida que procesa sentencias del lenguaje que define, construye el árbol sintáctico decorado y el grafo de dependencias entre los atributos. Por último, una vez que ha terminado el procesamiento, recorre el grafo de dependencias para evaluar el valor de cada una de las instancias de atributos de cada

nodo del árbol. Dicha evaluación se lleva a cabo *bajo demanda*, de forma que, para determinar el valor de un atributo, se determinan previamente los valores de los atributos de los que éste depende. Así mismo, una vez determinado el valor, se almacena para usos futuros. Nótese que el proceso debe requerir el valor de cada atributo del grafo, ya que el proceso de evaluación bajo demanda, requerido sobre los atributos sintetizados de la raíz, no tiene porqué garantizar la evaluación completa de todos los atributos. El Apéndice A muestra un ejemplo de generador de supuestos generado por el metagenerador.

3.4.1.4 Interfaz de usuario

La herramienta de autoría dispone de tres interfaces distintas en función de la tarea que se quiera llevar a cabo en el editor. Estas vistas son:

Figura 3.7. Captura del gestor de gramáticas

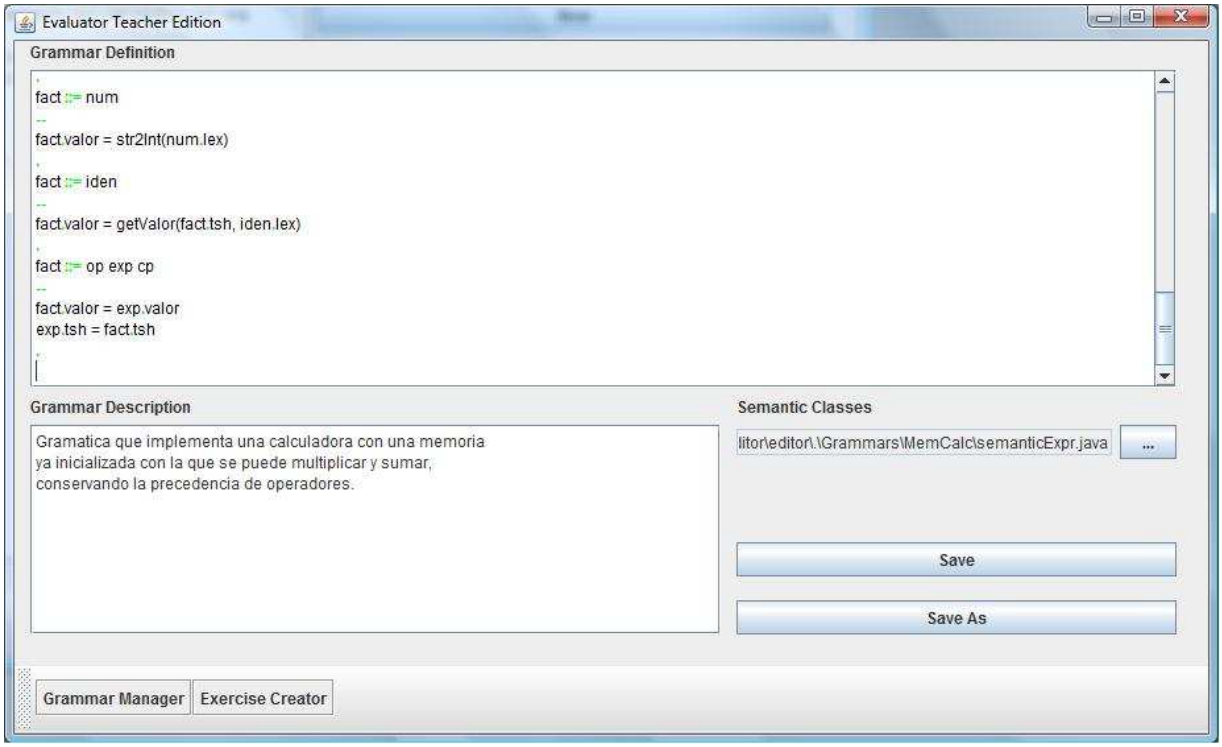


- El *gestor de gramáticas* es la primera ventana que se muestra al iniciar el editor. La ventana muestra las distintas gramáticas registradas en el sistema permitiendo borrarlas, abrir su especificación o compilarlas. Además permite crear una nueva gramática, lo que cambiará la vista actual a la vista del *editor de gramáticas*. Por último, la acción de compilar una gramática procesa la descripción de ésta y crea

el generador de supuestos para su posterior uso en la configuración de un ejercicio. La Figura 3.7 muestra una captura de la interfaz del gestor de gramáticas.

- El *editor de gramáticas* es la interfaz donde se especifican los lenguajes que se emplearán en la creación de supuestos para los ejercicios. Esta interfaz ofrece sendos editores para especificar el lenguaje d*e forma informal, fo*rmal (mediante el lenguaje descrito anteriormente) y especificar la ruta de la clase semántica que contendrá las funciones semánticas necesarias para el lenguaje. Como característica especial, el editor de gramáticas es sensible a la sintaxis resaltando en distintos colores las palabras reservadas. La Figura 3.8 muestra una captura del editor de gramáticas.

Figura 3.8. Captura del editor de gramáticas

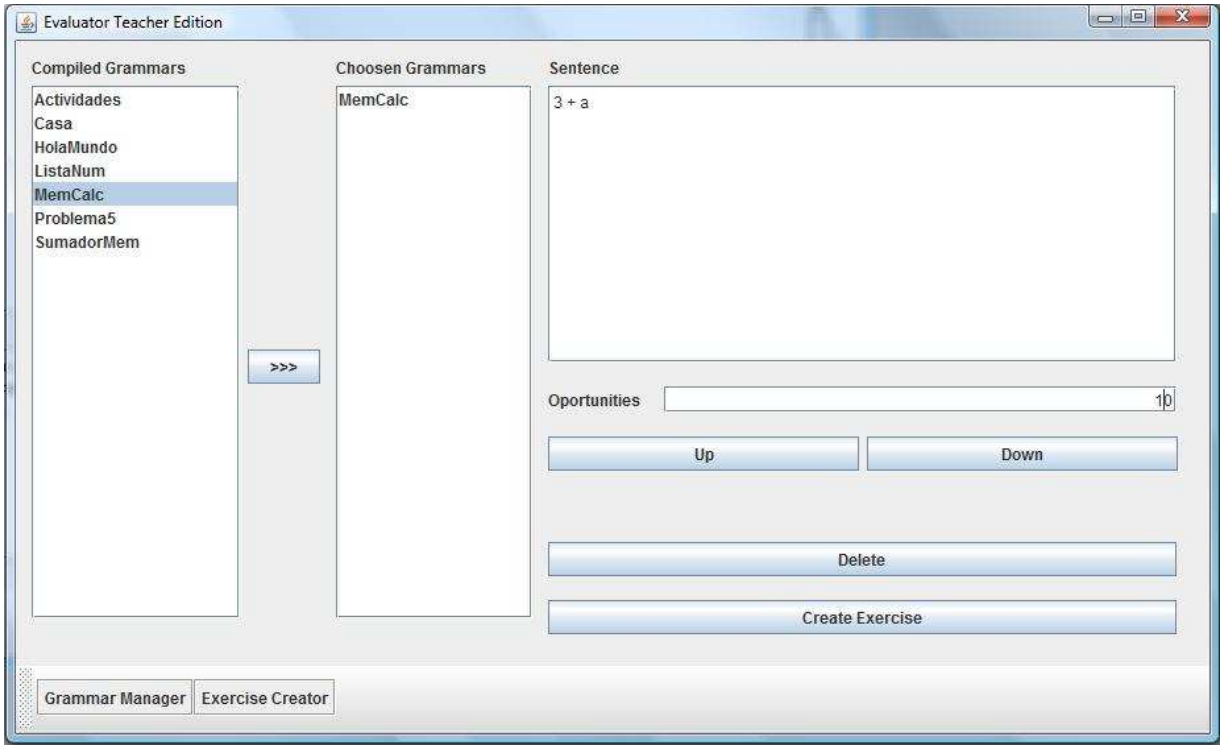


- El *creador de ejercicios* permite componer un ejercicio a través de los distintos supuestos escogidos por el *autor*. Para ello, la interfaz muestra los distintos generadores que han sido creados correctamente para que puedan ser escogidos para componer el ejercicio. Para cada uno de los generadores escogidos, el autor deberá especificar la sentencia y el número de errores máximos que compondrán

el supuesto. Por último, cuando el autor haya configurado cada uno de los supuestos que compondrán el ejercicio, podrá pulsar el botón de creación de ejercicios para codificar el ejercicio creado en un archivo XML de su elección. La Figura 3.9 muestra una captura de la interfaz del *creador de ejercicios*.

Con estas interfaces el autor es capaz de crear y mantener distintos lenguajes de forma sencilla y amigable. También permite crear ejercicios de forma rápida y sencilla, una vez se dispone de un repositorio grande de generadores de supuestos. Efectivamente a pesar de que la especificación de un lenguaje puede ser algo pesada, este trabajo se ve recompensado al poder utilizar el lenguaje varias veces en distintos ejercicios sin apenas esfuerzo.

Figura 3.9. Captura del creador de ejercicios



3.4.2 El Generador de Videojuegos

El generador de videojuegos es la herramienta encargada de transformar los ejercicios descritos acorde al modelo de ejercicio, en videojuegos donde cada supuesto del ejercicio se mapea en un mundo laberíntico, de acuerdo con el modelo de juego anteriormente introducido. De esta forma, cada supuesto constituye un nivel del juego, de tal forma que (ver la Figura 3.10):

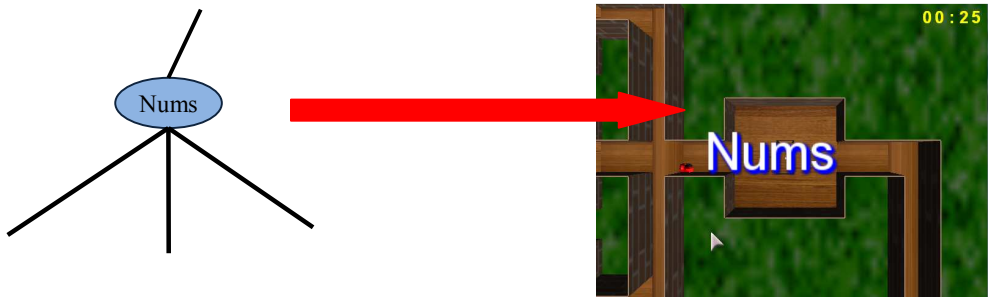
- Cada nodo del árbol se mapea en una habitación (Figura 3.10a).

- Las listas de hijos se mapean en pasillos que conectan la habitación asociada al padre con la asociada al hijo, pasando por un *hall* (Figura 3.10b).
- Cada atributo se mapea en una caja de la mesa contenida en dicha habitación (Figura 3.10c).
- Las dependencias entre atributos se utilizan para discriminar entre acciones de evaluación lícitas e ilícitas.
- Los valores de atributos, así como el texto correspondiente a las producciones y ecuaciones almacenados en los nodos no terminales, se utilizarán para proporcionar pistas a los estudiantes: los valores se mostrarán cuando estén completamente calculados los atributos (Figura 3.10d), mientras que los correspondientes fragmentos de las gramáticas de atributos se utilizarán para configurar *oráculos* (personajes que ofrecen pistas) situados en los halls (Figura 3.10e).

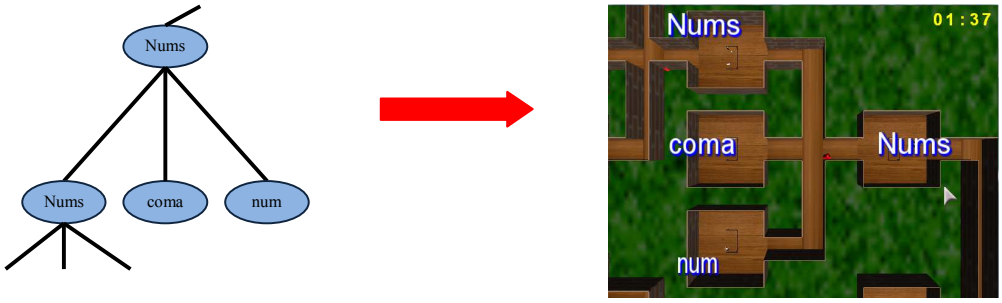
Desde un punto de vista técnico, el generador puede concebirse como una transformación entre el modelo del ejercicio y el modelo del juego. Dicha transformación es, en gran medida, directa. La mayor dificultad, quizá, estribe en generar una disposición atractiva de habitaciones y pasillos en el laberinto. Este problema se reduce a uno de diseño de un dibujo visualmente atractivo de árboles. En particular, la estrategia implementada en el generador, que mejora substancialmente la utilizada en [22], se basa en los algoritmos descritos en [51][74].Por ello se plantea un algoritmo donde se construye un mapa simbólico dividido en celdas y se van rellenando las celdas por niveles. Si en algún punto alguna celda es ocupada por más de una habitación, se reajusta la posición de la habitación padre ajustando la longitud del pasillo que une la habitación padre con su padre. A continuación se ilustra el mecanismo de este algoritmo con un sencillo ejemplo paso a paso.

Figura 3.10. Tabla comparativa entre distintos elementos de los árboles sintácticos decorados y los videojuegos generados

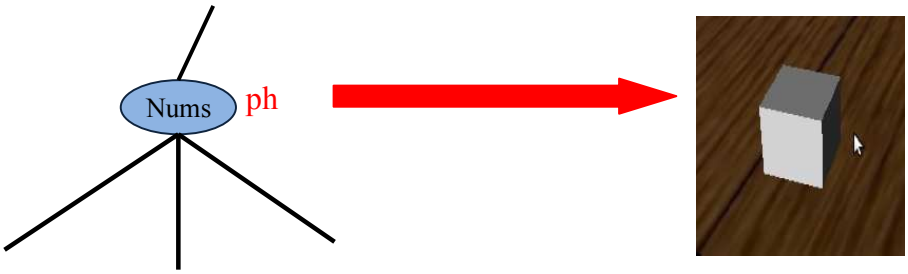
a)



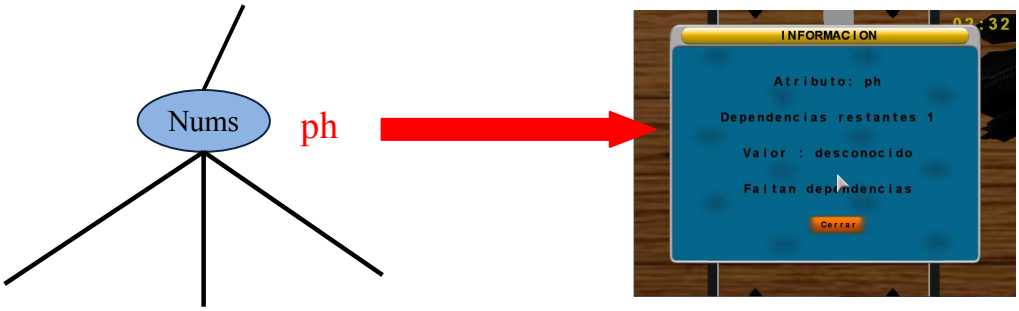
b)



c)



d)



e)

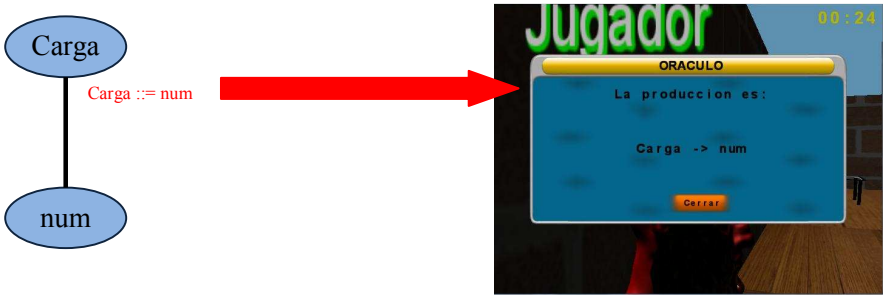
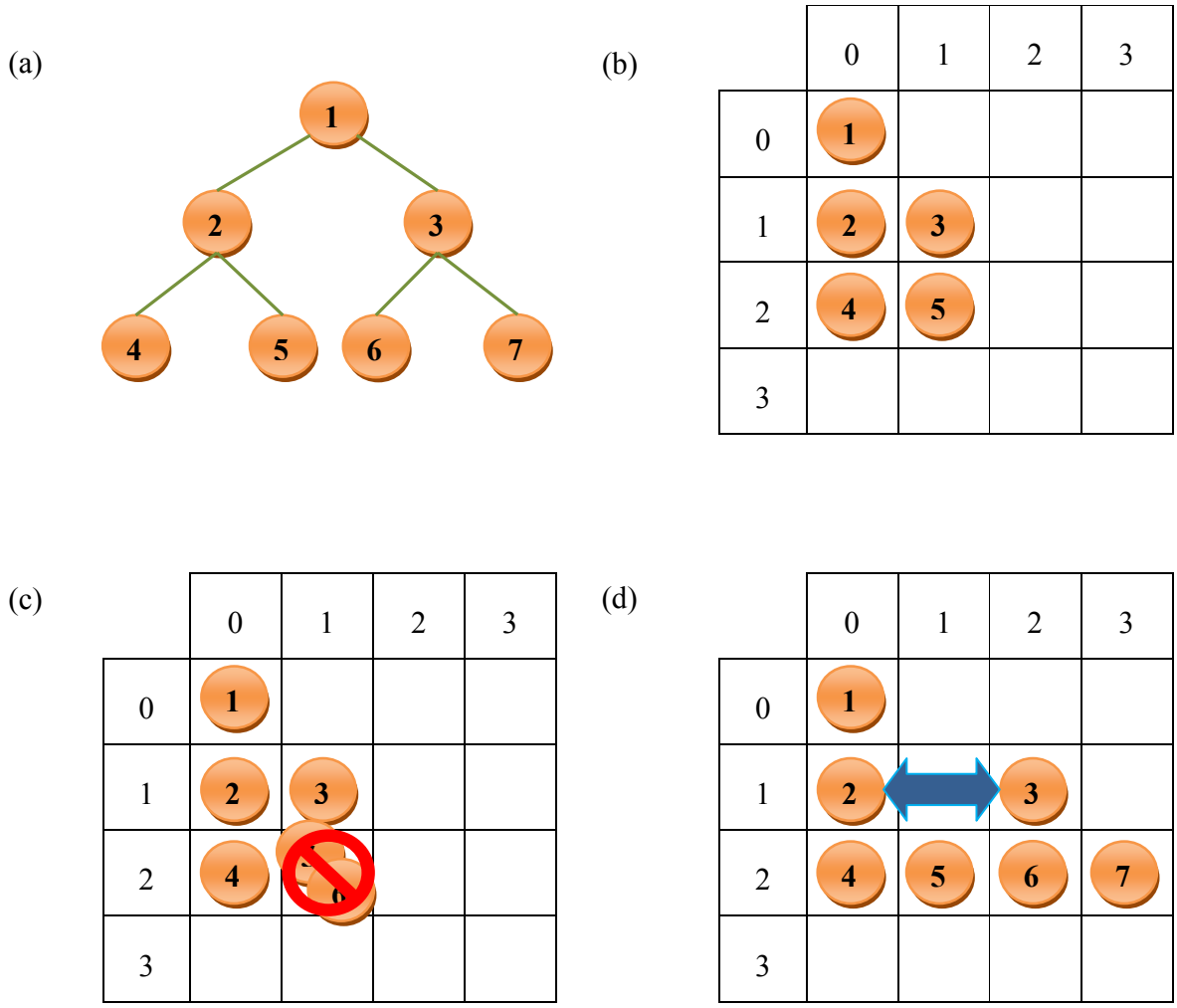


Figura 3.11. Esquema gráfico del funcionamiento del algoritmo de dibujado de laberintos



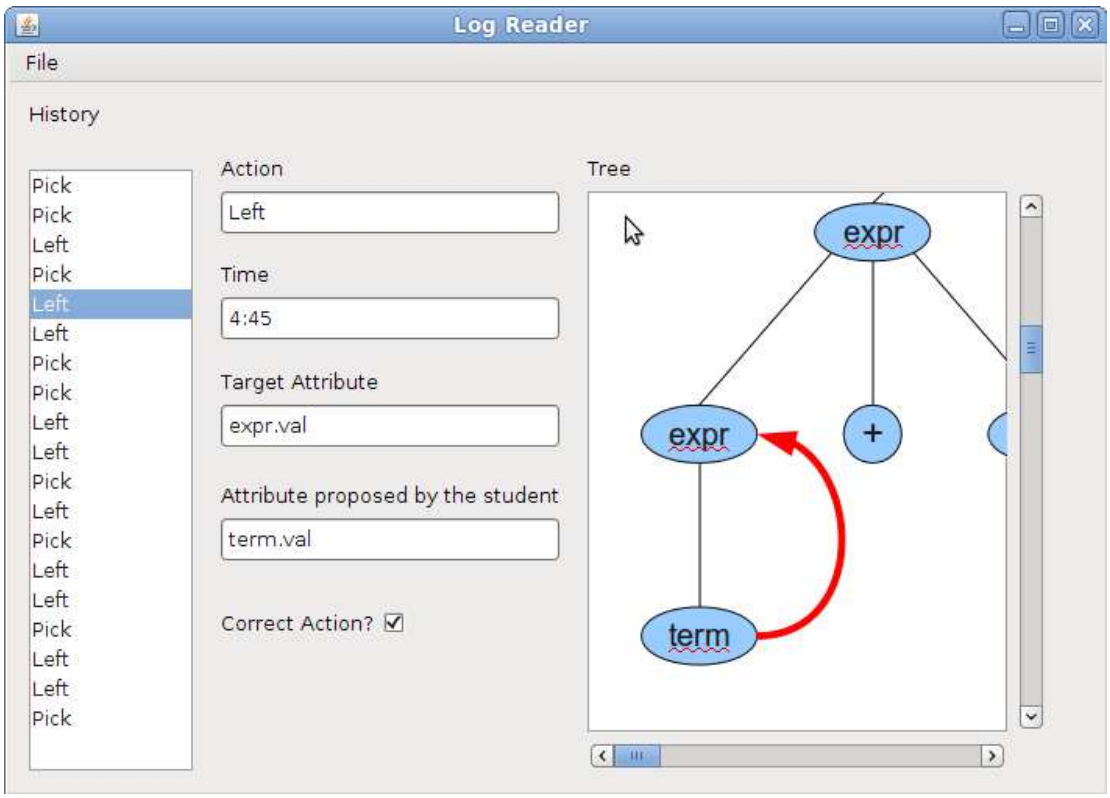
La Figura 3.11a muestra el árbol que se usará de ejemplo, que consta de tres niveles y en el que cada nodo tiene dos hijos. En una primera iteración se van colocando los nodos por niveles. La posición que ocupará el primer nodo hijo de un nodo padre será la misma que la del padre pero en el siguiente nivel. El resto de hijos ocuparán las siguientes posiciones (Figura 3.11b). A medida que se van llenando las casillas del mapa se puede detectar si dos nodos ocupan la misma casilla. Cuando ocurra esto, el algoritmo aumentará el espacio entre los nodos que estén al mismo nivel que el padre del nodo que provocó este conflicto. Este caso queda reflejado en la Figura 3.11c cuando el nodo 6 intenta ocupar la casilla que ocupa el nodo 5. Una vez reajustadas las distancias entre habitaciones se reconstruye el árbol desde el nivel en el que se ha reajustado la distancia entre habitaciones y el proceso continúa hasta encontrar algún otro

conflicto o terminar de explorar el árbol. La Figura 3.11d muestra el resultado final. Una vez terminado este proceso se transforman esas coordenadas discretas en las coordenadas finales que conformarán el laberinto en *Evaluators*.

3.4.3 La Herramienta de Análisis

Durante la ejecución del videojuego, se registran las acciones llevadas a cabo por el estudiante a lo largo de los niveles que éste juega. Las acciones que se registran son las relacionadas con la resolución del problema: coger el objeto de una caja y depositar un objeto en una caja (desde el punto de vista de las gramáticas de atributos, acciones de evaluación semántica). Además, en las operaciones en las que se depositan los objetos se anota si la acción fue de éxito o no. Estos *logs* se almacenan en un archivo XML para que pueda ser procesado posteriormente.

Figura 3.12. Concepto de diseño de la herramienta de análisis



Para procesar estos *logs* se emplea la herramienta de análisis. Esta herramienta muestra el árbol sintáctico y, paso a paso, las acciones que ha llevado a cabo el alumno sobre dicho árbol.

Actualmente esta herramienta se encuentra en una fase inicial de desarrollo, habiéndose construido únicamente algunas maquetas de la misma[7].

La Figura 3.12. muestra una captura de dicha maqueta, que ilustra el espíritu que se persigue con la construcción de dicha herramienta. Se observa que será posible visualizar el árbol sintáctico, ir navegando entre las acciones realizadas por el alumno y observar las consecuencias de éstas en el árbol sintáctico. Es notorio observar que el análisis se entiende en el dominio del modelo del ejercicio, y no en el dominio del modelo del juego, a fin de facilitar dicho análisis, que, en última instancia, conducirá a la evaluación del alumno.

3.5 Creación de Ejercicios con *Evaluators*

Como se ha discutido anteriormente, la herramienta de autoría de *Evaluators* facilita al docente la actividad de creación de los ejercicios. Para la creación de ejercicios el docente debe, en primer lugar, especificar las gramáticas de atributos que describirán los problemas de procesamiento de lenguajes que se usarán en el ejercicio, y después especificar las frases que producirán los árboles de análisis sintácticos decorados sobre los que el alumno tendrá que trabajar. Esto facilita la labor del docente, ya que podrá concentrarse en escribir gramáticas de atributos, y no en la tediosa tarea de describir árboles atribuidos, y, además, podrá reutilizar dichas gramáticas en la producción de una gran cantidad de ejercicios.

Para ilustrar el proceso de creación de un ejercicio, vamos a utilizar un pequeño lenguaje que especifica una calculadora que permite sumar y multiplicar números, teniendo en cuenta la precedencia de estas operaciones. Además, dispone de una memoria de constantes predefinidas. La Figura 3.13 muestra la gramática de atributos que especifica el problema de procesamiento de lenguaje explicado codificada con el lenguaje de especificación empleado en el editor de gramáticas.

[7] Actualmente, como paso previo a esta herramienta, se está utilizando un procesador de logs rudimentario que transforma los archivos a texto plano.

Figura 3.13. Gramática de atributos del lenguaje de la calculadora con la memoria

```
nt(exp,[mem_inh],[val]).
nt(term,[mem_inh],[val]).
nt(fact,[mem_inh],[val]).
nt(s,[],[val]).
axiom(s).

t(num,int).
t(iden,id).
t(add,lit("+")).
t(mul,lit("*")).
t(op,lit("(")).
t(cp,lit(")")).
t(end,lit("$")).

s ::= exp end
--
exp.mem_inh = initTable()
s.val = exp.val
.
exp ::= exp add term
--
exp.val = exp.val(1) + term.val
exp.mem_inh(1) = exp.mem_inh
term.mem_inh = exp.mem_inh
.
exp ::= term
--
exp.val = term.val
term.mem_inh = exp.mem_inh
.
term ::= term mul fact
--
term.val = term.val(1) * fact.val
term.mem_inh(1) = term.mem_inh
fact.mem_inh = term.mem_inh
.
term ::= fact
--
term.val = fact.val
fact.mem_inh = term.mem_inh
.
fact ::= num
--
fact.val = str2Int(num.lex)
.
fact ::= iden
--
fact.val = getValor(fact.mem_inh, iden.lex)
.
fact ::= op exp cp
--
fact.val = exp.val
exp.mem_inh = fact.mem_inh
        .
```

La herramienta de autoría permite codificar directamente dicha gramática de atributos. El metagenerador de supuestos permitirá, así mismo, compilar dicha gramática para producir un generador de supuestos adecuado (la Figura 3.14 muestra un fragmento del código CUP producido en dicho proceso).

Figura 3.14. Fragmento del código CUP generado por el *metagenerador de supuestos*.

```
117 // Declaración de Terminales
118 terminal Token num, cp, op, mul, add, iden, end;
119 
120 // Declaración de no terminales
121 non terminal NodoNT exp;
122 non terminal NodoNT fact;
123 non terminal NodoNT s;
124 non terminal NodoNT term;
125 
126 
127 //Reglas de la gramática
128 
129 s ::= exp : _ref_exp0 end : _ref_end0{:
130     RESULT = new NodoNT(""+parser.id, "s", null);
131     parser.id++;    Atributo s_val = new Atributo("val", RESULT);
132     s_val.setDependeDe(new ArrayList<Atributo>());
133     parser.depTable.put(s_val, new Stack<Object>());
134     ArrayList<Nodo> childs = new ArrayList<Nodo>(); childs.add(_ref_exp0);
135     _ref_exp0.setPadre(RESULT);
136     RESULT.setHijos(childs);
137     NodoT terminal1 = new NodoT(""+parser.id, "end", null);
138     parser.id++;    childs.add(terminal1);
139     terminal1.setPadre(RESULT);
140     Atributo lex_end1 = new Atributo("lex", terminal1);
141     lex_end1.setValor("\""+_ref_end0.getLex()+"\"");
142     lex_end1.setDependeDe(new ArrayList<Atributo>());
143     parser.depTable.put(lex_end1, new Stack<Object>());
144     parser.depTable.get(lex_end1).push(_ref_end0.getLex());
145     RESULT.setHijos(childs);
146     ArrayList<Atributo> dep = null;
147     Nodo nodo = null;
148     Atributo attr = null;
149     Stack<Object> pila = null;
150     nodo = RESULT;
151     attr = s_val;
152     dep = s_val.getDependeDe();
153     pila = parser.depTable.get(attr);
154     pila.push(parser.getAtributo(_ref_exp0, "val"));
```

Una vez producido este generador de supuestos, podrá ser empleado en la construcción de ejercicios. Para este ejemplo vamos a construir un ejercicio que sólo tenga un nivel, el cual consistirá en una frase del lenguaje soportado por la anterior gramática. La frase que emplearemos es *"3+(5*a)$"*. Una vez hecho esto, la herramienta generará automáticamente el ejercicio, y toda la información quedará codificada en un archivo XML (la Figura 3.15 muestra un fragmento del archivo XML generado).

Figura 3.15. Fragmento del archivo XML que contiene la codificación del ejercicio

```
<?xml version="1.0" encoding="UTF-8"?>
<java version="1.6.0_24" class="java.beans.XMLDecoder">
 <object class="Modelo.Ejercicio">
  <void property="grammar">
   <string>#fact -&gt; num
!fact(0).valor = str2Int(num(0).lex)
#fact -&gt; iden
!fact(0).valor = getValor(fact(0).tsh ,iden(0).lex)
#fact -&gt; op exp cp
!exp(0).tsh = fact(0).tsh
!fact(0).valor = exp(0).valor
#term -&gt; term mul fact
!term(0).valor = def_MUL(term(1).valor ,fact(0).valor)
!term(1).tsh = term(0).tsh
!fact(0).tsh = term(0).tsh
#term -&gt; fact
!term(0).valor = fact(0).valor
!fact(0).tsh = term(0).tsh
#s -&gt; exp fin
!s(0).valor = exp(0).valor
!exp(0).tsh = initTable()
#exp -&gt; exp add term
!exp(1).tsh = exp(0).tsh
!exp(0).valor = def_ADD(exp(1).valor ,term(0).valor)
!term(0).tsh = exp(0).tsh
#exp -&gt; term
!exp(0).valor = term(0).valor
!term(0).tsh = exp(0).tsh
</string>
  </void>
  <void property="maxErrors">
   <int>10</int>
  </void>
  <void property="name">
   <string>0</string>
  </void>
  <void property="sentence">
   <string>Gramatica que implementa una calculadora con una memoria
ya inicializada con la que se puede multiplicar y sumar,
conservando la precedencia de operadores.
```

3.6 Generación de Videojuegos en *Evaluators*

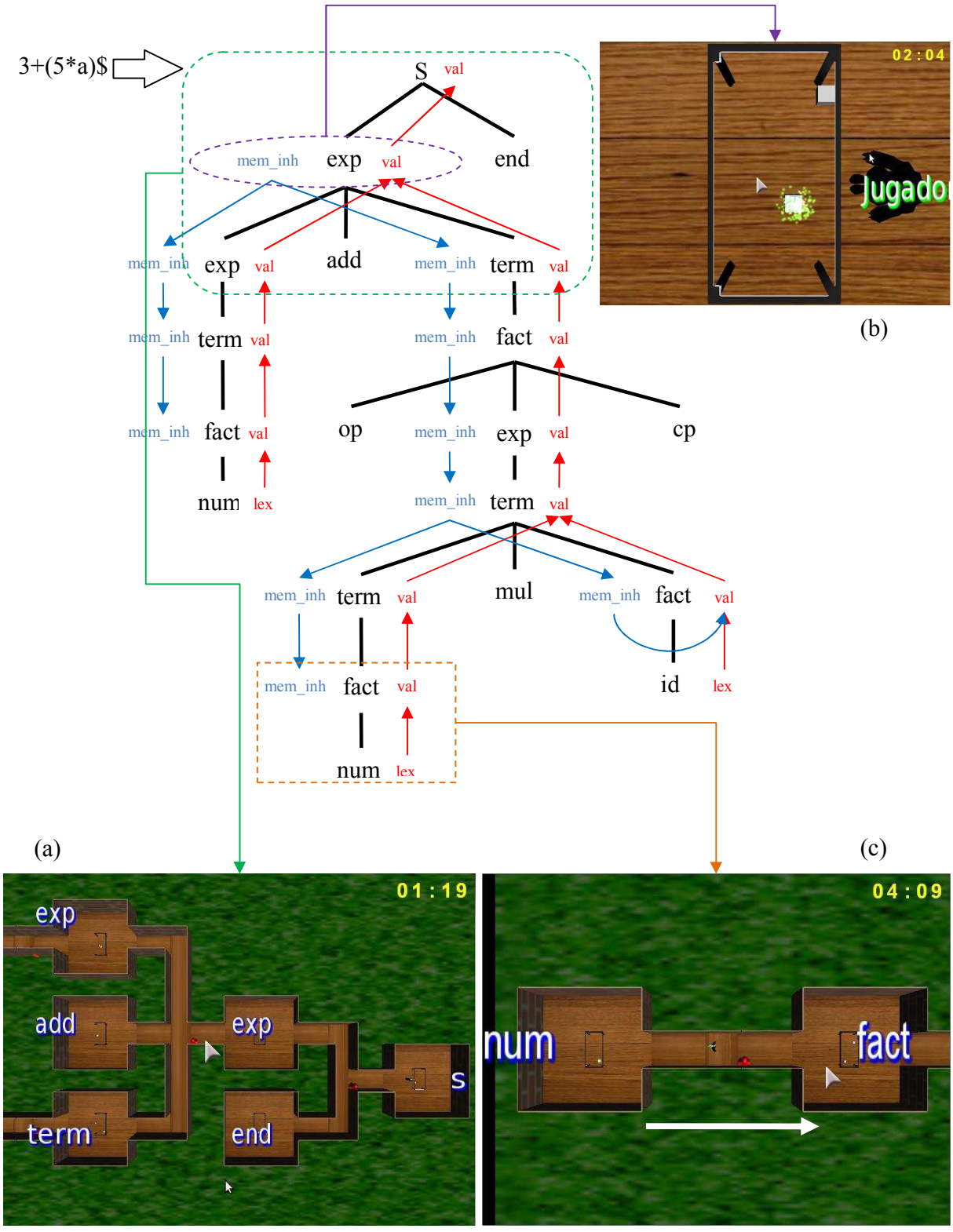
El videojuego se genera con el generador de videojuegos a partir de los ejercicios creados por el docente con la herramienta de autoría y codificados en XML. El generador procesa el ejercicio y genera los distintos niveles mapeando cada árbol sintáctico decorado en un laberinto siguiendo los convenios descritos en el punto 3.4.2 .

Siguiendo con el ejemplo descrito anteriormente, se ilustrarán las transformaciones entre el modelo del ejercicio y el modelo del juego llevadas a cabo en la generación del videojuego. En

la sección anterior el docente había creado un ejercicio con un lenguaje que describía una calculadora muy sencilla con una memoria que contenía una seria de constantes predefinidas. También había escogido una frase de ese lenguaje y había codificado en un archivo XML el ejercicio. Para este lenguaje y la sentencia escogida por el docente, el árbol sintáctico decorado es el representado en la Figura 3.16, donde se pueden apreciar las relaciones de dependencia entre los atributos de los distintos nodos:

- Se observa cómo el subárbol sintáctico remarcado en verde guarda relación con la parte de laberinto mostrado en la Figura 3.16a. Se observa también cómo las habitaciones se corresponden con cada uno de los nodos marcados en el rectángulo verde y cada uno de los pasillos que unen estas habitaciones son una forma metafórica de representar los arcos que unen cada uno de los nodos en el árbol sintáctico.
- Dentro de cada habitación, tal como se muestra en la Figura 3.16b, existe una mesa que contiene cajas. Esas cajas corresponden a instancias de atributos pertenecientes al nodo que representa la habitación. El recuadro lila en la Figura 3.16 muestra la correspondencia entre el nodo, junto con sus instancias de atributo, y la habitación con su mesa donde se encuentran las cajas que representan a dichas instancias de atributo. Por último, para que el usuario consiga completar el nivel completamente deberá resolver todas las dependencias del árbol sintáctico. Estas dependencias determinan, por tanto, los movimientos lícitos e ilícitos dentro del juego.

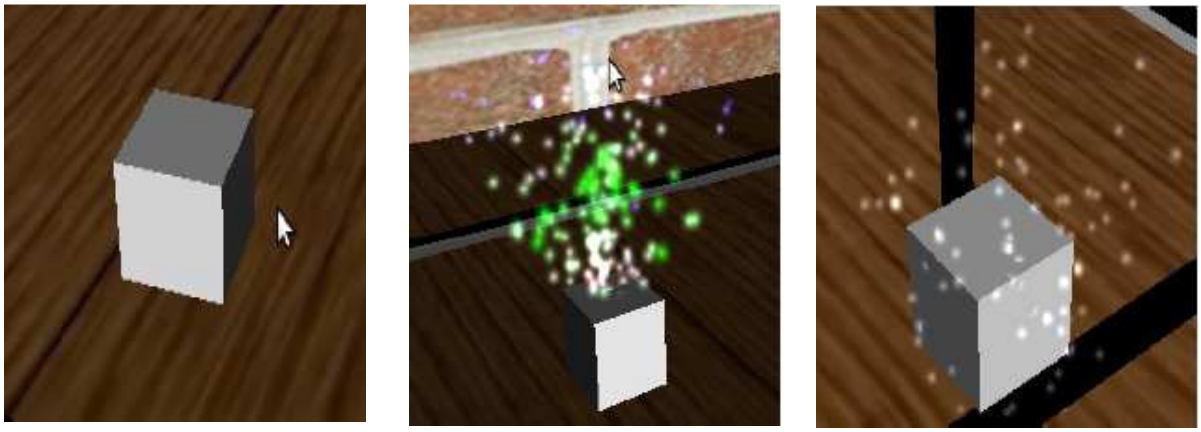
Figura 3.16. Comparación de distintos aspectos de *Evaluators* con un árbol sintáctico decorado.



3.7 Uso de videojuegos en *Evaluators*

Una vez generados, los juegos permiten a los estudiantes emular el proceso de evaluación semántica en cada nivel. Efectivamente, la misión de los estudiantes es completar cada uno de los niveles representados por los supuestos propuestos por el docente, lo que supone resolver correctamente la evaluación semántica del problema que plantea cada supuesto. Para poder completar esta tarea, y siguiendo la metáfora descrita en la sección 3.3 , el estudiante debe proporcionar a cada caja los objetos adecuados de otras cajas. Desde la perspectiva de las gramáticas de atributos, esto es equivalente a decidir, para cada uno de los atributos desconocidos, los atributos que son necesarios para evaluarlos. Para hacer este proceso posible, *Evaluators* proporciona a los alumnos ayuda en forma de pistas:

Figura 3.17. Capturas de los distintos estados de las cajas de *Evaluators*

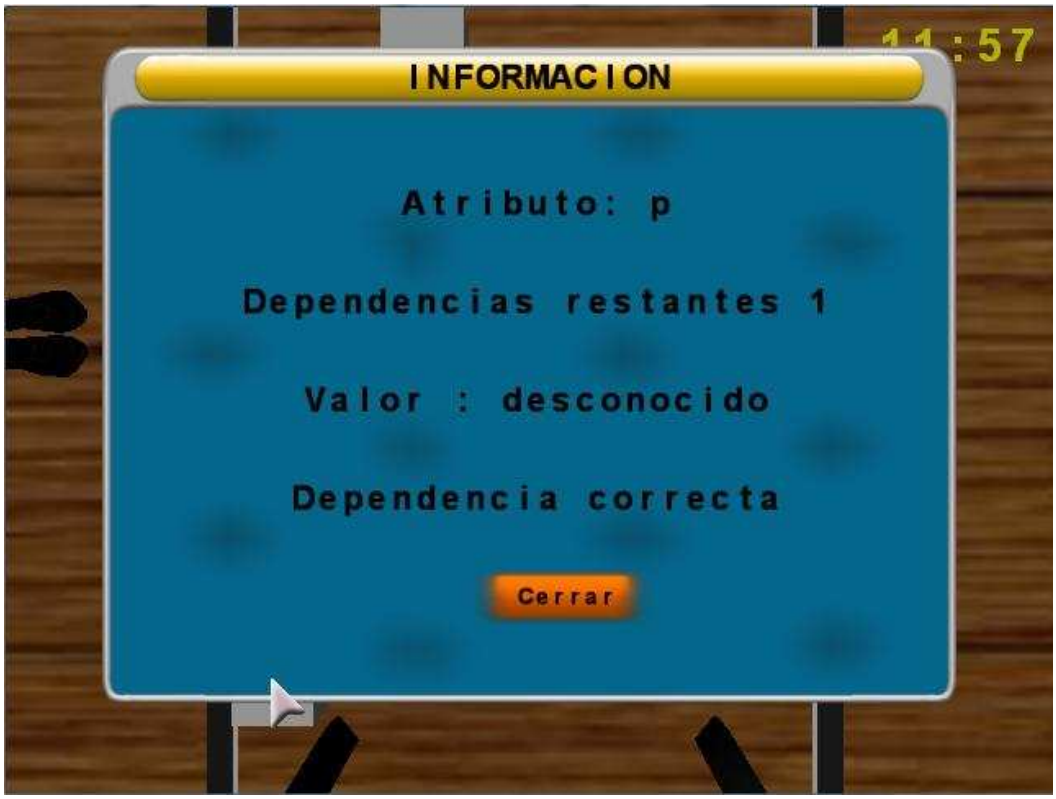


(a) (b) (c)

- Las cajas que corresponden con atributos cuyo valor es desconocido aparecen como inactivas (véase la Figura 3.17a). Por otro lado, las cajas correspondientes con atributos cuyo valor ya ha sido computado emiten un chorro de luces multicolores (véase la Figura 3.17b).
- Los errores de los estudiantes se corresponden con depositar un objeto en la caja equivocada. En este caso el sistema muestra una pequeña columna de humo gris que sale de la caja donde se deposita el objeto (véase la Figura 3.17c). Por otro lado, si el estudiante deposita un objeto en la caja adecuada, el efecto dependerá de si el proceso de evaluación del atributo asociado a la caja ha sido completado o

no. En el primer caso, la caja emitirá el chorro de luces multicolores anteriormente mencionado. En el otro caso, la caja permanecerá inactiva.

Figura 3.18. Captura de un panel de información asociado a una caja.



Además de estas pistas, el sistema proporciona un panel (véase la Figura 3.18) con información sobre el estado de la caja cada vez que el usuario deposita un objeto en una caja o cuando el usuario así lo solicita. La información mostrada es la siguiente: el nombre del atributo al que representa la caja, número de referencias que aún no han sido resueltas, el valor del objeto, si ya ha sido computado, y un campo donde muestra si el ultimo objeto depositado fue correcto o no.

Los videojuegos proporcionan también una vista aérea del laberinto para ayudar al alumno a orientarse por el mundo (véase la Figura 3.19a), una pantalla donde el alumno puede consultar la especificación del ejercicio (véase la Figura 3.19b) y unos oráculos situados en los pasillos donde el alumno puede consultar qué producción es la asociada a ese pasillo (véase la Figura 3.20).

Figura 3.19. Capturas de la vista aérea y del enunciado de *Evaluators*.

(a) (b)

Figura 3.20. Un oráculo dentro de un laberinto de *Evaluators* y la información que muestra.

Finalmente, para almacenar los objetos (los valores de atributos), el sistema incluye el concepto de *inventario*. Este inventario consiste en una lista de objetos (valores de atributos). Con el fin de orientar a los alumnos, cuando un objeto del inventario es seleccionado, la cámara del juego se sitúa encima de la habitación de donde se recogió ese objeto usando la vista aérea (véase la Figura 3.21).

Figura 3.21. Captura del inventario en *Evaluators*.

3.8 Evaluación del Estudiante en *Evaluators*

La evaluación del estudiante se lleva a cabo cuando los alumnos han terminado de jugar con el videojuego generado con *Evaluators*. Los alumnos deberán entregar a los docentes los archivos XML donde se han registrado todas las acciones que han llevado a cabo estos durante el tiempo que han jugado al ejercicio. El docente puede entonces evaluar las acciones que han realizado los alumnos en el videojuego, pero traduciéndolas en términos de las dependencias que han propuesto entre las instancias de los atributos de cada nodo del árbol sintáctico. Con esta evaluación el docente puede ser consciente de los principales problemas que encuentran sus alumnos en determinados aspectos del formalismo de las gramáticas de atributos e intentar solventarlos en clase o con ejercicios de refuerzo.

3.9 A Modo de Conclusión

A lo largo de este capítulo se ha desarrollado una propuesta para paliar las distintas carencias y deficiencias observadas en el Capítulo 2. De esta forma:

- En primer lugar, se ha descrito el modelo de proceso de producción seguido para el desarrollo de un sistema educativo basado en videojuegos, comentando sus actores, actividades y productos.
- Este modelo de proceso se ha aplicado al desarrollo de *Evaluators.* Se ha descrito la concepción de *Evaluators,* explicando por un lado los ejercicios que se soportarán mediante los videojuegos y cómo se transformarán estos al videojuego adoptando una metáfora adecuada. Después, se ha descrito el conjunto de herramientas que compondrán el sistema: la herramienta de autoría, el generador de videojuegos y la herramienta de análisis. Estas herramientas tienen como función, respectivamente, editar y componer los ejercicios, transformar la especificación de los ejercicios en videojuegos acorde a la metáfora explicada, y analizar el comportamiento de los estudiantes durante la realización de los ejercicios. Seguidamente, se explica el uso de la herramienta de autoría para crear los ejercicios que los alumnos resolverán jugando con *Evaluators*. Es de interés apuntar que la herramienta de autoría permite a los docentes crear de forma sencilla y amigable gran cantidad de ejercicios con tan solo un pequeño esfuerzo inicial. A continuación, se explica los distintos procesos que se llevan a cabo para generar los videojuegos, y más concretamente la transformación de la especificación de un ejercicio en un videojuego. Por último, se comenta brevemente el proceso de evaluación de los estudiantes llevado a cabo por los docentes, haciendo uso de la herramienta de análisis.

De esta forma, se ha presentado un modelo de proceso viable, sencillo y eficaz para la creación de sistemas educativos dirigidos por ejercicios, y dicho modelo de proceso se ha aplicado de forma efectiva para la creación de un sistema educativo. Este sistema es un generador de videojuegos educativos para la enseñanza de las gramáticas de atributos, enfocado a los aspectos de especificación, en lugar de a los aspectos de implementación (tendencia mayoritaria en las herramientas examinadas en el capítulo anterior), y centrado en mejorar la experiencia educativa en fases tempranas de la enseñanza de las materias de procesadores de

lenguaje (en lugar de en conceptos y procedimientos relativos a fases más avanzadas, como ocurre con las herramientas examinadas en el capítulo anterior). El modelo de proceso ha permitido, así mismo, contemplar conjuntamente todos los aspectos del sistema (herramientas de autoría, herramientas de análisis, etc.), a fin de facilitar su aplicación práctica. Efectivamente, una particularidad de este modelo de proceso es que no se detiene cuando el producto (o productos) final(es) se ha(n) terminado, sino que permite evolucionar y mejorar éste (estos) a lo largo del tiempo, en base a resultados obtenidos a través de procesos de evaluación formativa, consiguiendo así un sistema completo y de calidad. El próximo capítulo evidencia este carácter incremental de la propuesta, describiendo las evaluaciones preliminares de (una versión previa de) el sistema realizadas, y cómo dichas evaluaciones ha contribuido a mejorar el mismo, dando lugar a los distintos componentes presentados en este capítulo.

Capítulo 4. Evaluación

4.1 Introducción

Tal y como se ha descrito en el capítulo anterior, el modelo de proceso introducido contempla explícitamente la realización de evaluaciones formativas. Dichas evaluaciones pueden ser tanto pedagógicas como técnicas, y abordar tanto los videojuegos educativos como las herramientas utilizadas en la creación de los mismos. La finalidad de estas evaluaciones es mejorar los sistemas de enseñanza / aprendizaje basados en juegos. Efectivamente, dichas evaluaciones permiten encontrar y solucionar posibles fallos y/o malfuncionamientos. También se pretende con estas evaluaciones impulsar la calidad del software desarrollado para alcanzar el máximo de calidad tanto educativa como lúdica. Por último, pero no por ello menos importante, estas evaluaciones también se realizan para medir la eficacia y eficiencia del sistema en relación con el progreso de los alumnos, a fin de mejorar el mismo para mejorar dicha eficacia y eficiencia.

En este capítulo se describe una evaluación preliminar llevada a cabo en este sentido durante el curso 2010-2011 en el contexto de la asignatura de Procesadores de Lenguajes impartida en la Universidad Complutense de Madrid. Los objetivos de dicha evaluación fueron los siguientes:

- Evaluar la adecuación de las herramientas de autoría, generación de videojuegos y análisis a las necesidades de los docentes.
- Contrastar la eficacia educativa del nuevo método y compararla con las estrategias convencionales de resolución del tipo de ejercicios abordado.
- Evaluar el grado de satisfacción de los alumnos con el nuevo método de enseñanza.

La evaluación se realizó utilizando una versión preliminar de *Evaluators*, basada en el sistema descrito en [22]. Los resultados de la misma se utilizaron para producir la versión actual del sistema, descrita en el capítulo anterior. A continuación se describen los distintos aspectos de dicha evaluación. La sección 4.2 describe brevemente las características de la versión de *Evaluators* tomada como punto de partida para la experiencia. La sección 4.3 describe la evaluación preliminar con docentes. Las secciones 4.4 y 4.5 describen la evaluación preliminar con alumnos.

4.2 El sistema de partida

Evaluators parte, en su versión inicial, como un proyecto de Sistemas Informáticos [22]. En esta primera versión, el sistema era capaz de cargar ejercicios que consistían en la resolución de un solo supuesto. Además, con el fin de aumentar la componente lúdica del videojuego se disponían una serie de trampas a lo largo del laberinto, que se presentaban al alumno, para que éste las esquivase. En cuanto a los consejos y pistas que se ofrecían al alumno, se disponía de información de los objetos registrados en el inventario y las señales luminosas de cada una de las cajas que se encontraban en las mesas.

En cuanto a las capacidades de esta primera versión, éstas se reducían a un editor de árboles sintácticos decorados, donde el usuario debía definir la estructura del árbol sintáctico del supuesto, junto con sus atributos y las dependencias entre estos.

4.3 Evaluación de la adecuación de las herramientas

Con el fin de evaluar y medir la calidad de las herramientas incluidas en la versión preliminar de *Evaluators,* se planteó un estudio a pequeña escala con docentes, de naturaleza cualitativa e informal. A continuación se detalla dicha evaluación.

4.3.1 Descripción de la experiencia

Para esta experiencia se contó con la presencia de tres docentes:

- Un docente de las materias impartidas en la asignatura de Procesadores de Lenguaje (**D1**).
- Una docente de Informática, ajena a la materia de Procesadores de Lenguaje, pero experta en el desarrollo de herramientas educativas para la enseñanza de la informática (**D2**).
- El propio autor de este trabajo de investigación: becario de colaboración en el Departamento de Ingeniería del Software e Inteligencia Artificial (**D3**).

Los docentes **D1** y **D2** tuvieron acceso al sistema y a la documentación del mismo, y llevaron a cabo una evaluación informal exhaustiva de este material, orientada a la detección temprana de aspectos mejorables desde el punto de vista educativo. Por su parte, el docente **D3** se centró en la aplicación de la herramienta de autoría a varios casos de estudio complejos, a fin de determinar su usabilidad y aplicabilidad práctica. Por último, los tres participantes

contrastaron los resultados obtenidos, y propusieron una serie de mejoras sobre el sistema, que fueron llevadas a cabo.

4.3.2 Recogida de datos

La recogida de datos se llevó a cabo informalmente, en forma de registros y notas elaboradas por cada uno de los participantes.

4.3.3 Resultados

El modelo de videojuego obtuvo muy buenas opiniones, tanto por parte del docente **D1** (que ya había dirigido el desarrollo preliminar del mismo, pero que aún no había realizado un análisis exhaustivo de su potencial didáctico), como por parte de la docente **D2**, quien resaltó sobre todo la naturalidad de la metáfora adoptada. Como factores a mejorar, se expusieron los siguientes:

- Posibilidad de crear ejercicios encadenando varios supuestos, que deberían ser presentados en el videojuego como distintos niveles a superar por el usuario.
- Conteo del número de errores cometidos por el usuario, y reinicio del nivel según un número de errores especificados por el autor del ejercicio.
- Información adicional asociada a cada caja del laberinto, mostrando el nombre del atributo, el número de dependencias sin resolver, el valor del atributo (si está disponible) y el resultado de la última dependencia propuesta por el alumno.
- Posibilidad de mostrar, en todo punto, la descripción formal e informal del supuesto y la sentencia asociada a dicho supuesto.
- Creación de *logs*, donde se almacenan las acciones del usuario para su posterior análisis, y necesidad de una herramienta para analizar dichos *logs*.
- Un modelo de inventario más intuitivo, en el que sea posible localizar, en el mapa del laberinto, la procedencia de los distintos objetos.
- Eliminación de las trampas, y substitución de las mismas por *oráculos.* Efectivamente, la docente **D2** apuntó que las trampas, siendo su componente educativa nula, podían distraer también al usuario del objetivo final de resolver el ejercicio propuesto. En base a esta crítica, se estimó oportuno eliminar estos

artefactos, y substituirlos por oráculos, que, al ser consultados, informaran del fragmento de gramática aplicable en cada *hall*.

En cuanto a la herramienta de autoría, la experiencia llevada a cabo por el docente **D3** demostró su falta de utilidad práctica a la hora de crear ejercicios complejos. Efectivamente, **D3** comprobó cómo esta tarea podía resultar harto pesada y compleja, y además era muy fácil cometer errores en la especificación de un supuesto. El contraste de esta dificultad condujo a proponer el enfoque generativo aplicado en la actual versión de *Evaluators*, y descrito en el capítulo anterior.

4.3.4 Acciones emprendidas

En base a esta evaluación preliminar con docentes se llevó a cabo una reestructuración completa del sistema. El resultado fue, salvo ciertos detalles relativos a los videojuegos, el descrito en el capítulo anterior.

El principal avance de la reestructuración llevada a cabo está, sin duda alguna, en el nuevo enfoque de autoría adoptado, enfoque que, como ya se ha indicado, permite crear los supuestos a partir de las especificaciones. Dicha herramienta fue testada de nuevo por **D3**, quien pudo contrastar el espectacular aumento en la facilidad de uso y en la aplicabilidad práctica del nuevo enfoque.

4.4 Evaluación de la eficacia educativa

Durante esta evaluación se midió la eficacia educativa del sistema *Evaluators* con alumnos de la asignatura de Procesadores de Lenguaje que se imparte en la Facultad de Informática de la Universidad Complutense de Madrid. Las siguientes susbsecciones detallan la evaluación llevada a cabo.

4.4.1 Descripción de la experiencia

La experiencia consistió en la realización de cinco ejercicios, de dificultad ascendente, por parte de los alumnos de la asignatura de Procesadores de Lenguaje. Con el fin de medir la calidad del sistema educativo *Evaluators*, los alumnos se dividieron en dos grupos.

- Un primer grupo (grupo *experimental*) realizó los ejercicios utilizando *Evaluators*.

- Los alumnos restantes (grupo de *control*) realizaron los ejercicios en clase, en papel.

Para llevar a cabo la formación de grupos se solicitó participación voluntaria por parte de los alumnos, incentivada por un incremento en 0.5 puntos sobre la nota final del examen de Junio. Se entregó una encuesta a todos los alumnos participantes, a fin de recabar información, que se utilizó para garantizar la homogeneidad de los grupos resultantes.

Una vez formados los grupos, los alumnos llevaron a cabo la resolución de los ejercicios, contando para ello con dos sesiones tutorizadas de una hora cada una, y pudiendo completar dichos ejercicios en clase.

Finalmente, se realizó un test a todos los participantes, orientado a medir su grado de asimilación de los conceptos y procesos involucrados en los ejercicios propuestos.

4.4.2 Recogida de datos

Durante esta experiencia se utilizaron los siguientes instrumentos de recogida de datos:

- La encuesta inicial. Esta encuesta permitió recoger las calificaciones obtenidas por los participantes en materias anteriores relacionadas con las materias de Procesadores de Lenguaje, discriminar qué alumnos eran o no repetidores, el grado de conocimiento subjetivo de cada alumno en los distintos aspectos de la materia, y el gusto de los mismos por los videojuegos. Esta encuesta se incluye en el Apéndice B.
- Los ejercicios. Los ejercicios abordados fueron idénticos en ambos grupos. En el caso del grupo de control, se proporcionó estos ejercicios en papel (ver Apéndice C), mientras que en el caso del grupo experimental se proporcionó el correspondiente juego generado a partir de los mismos.
- El test final. Dicho test consistió en una secuencia de 21 preguntas *verdadero – falso* sobre conceptos relativos a estructura estática y modelo de procesamiento de las gramáticas de atributos. Dicho test se incluye en el Apéndice D.

4.4.3 Ejecución de la experiencia

La ejecución de la experiencia fue como sigue:

- Se comenzó aplicando la encuesta inicial, y ésta se utilizó para formar los grupos. El grupo experimental incluyó a 19 alumnos, y el de control a 26. El desequilibrio en número de participantes entre los dos grupos se debió a una razón logística: la capacidad de un laboratorio sólo permite 20 alumnos, uno por puesto. No obstante, en el diseño de los grupos se siguieron criterios para garantizar la homogeneidad, tanto en lo que se refiere a repetidores y a las actitudes hacia los videojuegos. Efectivamente, las proporciones de *no repetidores* vs. *repetidores* y *aficionados a videojuegos* vs. *no aficionados* se mantuvieron prácticamente idénticas en ambos grupos. Por lo demás, los alumnos se asignaron a los grupos de manera aleatoria, a fin de mantener la homogeneidad en calificación global y conocimientos previos.
- Seguidamente se procedió a aplicar los tratamientos. Como se ha indicado anteriormente, se destinaron dos sesiones de una hora, donde los alumnos debían resolver los cinco ejercicios propuestos. La estructura que se siguió fue similar en ambos grupos. Los docentes de ambos grupos emplearon parte de la primera sesión para explicar cómo se debían resolver los ejercicios. Para ello realizaron el primer ejercicio a modo de guía para los alumnos y después permitieron a los alumnos realizar el resto de ejercicios. La única diferencia entre ambos grupos es que el grupo del laboratorio debió realizar los ejercicios empleando *Evaluators,* y el otro grupo debió realizarlos por el método tradicional. Una vez finalizada la experiencia se recogieron las soluciones de cada alumno. En el laboratorio se recogieron los *logs* generados por los alumnos, y en la clase se recogieron las soluciones en papel.
- Por último, en una tercera sesión de una hora, todos los participantes realizaron el test final.

4.4.4 Resultados

La Tabla 4.1 muestra los resultados obtenidos organizados por los grupos en los que se separaron a los alumnos. La Tabla 4.1 contempla dos datos, la media de aciertos y la desviación típica entre éstos. Se aprecia que la media de aciertos entre ambos grupos no presenta grandes cambios, siendo ligeramente superior la media de aciertos del grupo que realizo los ejercicios

empleando el método tradicional (Grupo 1 en la tabla) frente a la que empleo *Evaluators* (Grupo 2 en la tabla). En resumen, estos resultados apuntan a que no hay diferencias significativas entre *Evaluators* y el método tradicional, lo cual supone que no hay un demérito de calidad entre el método propuesto con el método tradicional.

Tabla 4.1. Resumen de los resultados obtenidos en el post-test.

Grupo 1	Media de aciertos	14,58
	Desviación típica	2,19
Grupo 2	Media de aciertos	14,05
	Desviación típica	2,16

Se observa también cómo la desviación típica entre los dos grupos es muy similar, lo que refuerza la conclusión anterior.

4.5 Evaluación de la satisfacción de los alumnos

Durante esta evaluación se pretendió medir la opinión y grado de satisfacción de los alumnos. Para ello, se recogieron y analizaron las opiniones de los alumnos que utilizaron *Evaluators* durante la experiencia descrita en la sección previa. A continuación se detalla esta evaluación.

4.5.1 Descripción de la experiencia

La experiencia fue llevada a cabo de forma paralela al estudio explicado en la sección anterior. Los 19 alumnos que trabajaron, en la experiencia anterior, con los cinco ejercicios propuestos y que los resolvieron usando *Evaluators* fueron encuestados para obtener su valoración y grado de satisfacción con la herramienta.

4.5.2 Recogida de datos

Para recoger las opiniones de los alumnos se utilizó un cuestionario que consistía en diecisiete preguntas cuyas posibles respuestas estaban basadas en la escala Likert. Las preguntas realizadas se pueden encontrar en la Tabla 4.2. Las preguntas se elaboraron basándose y adaptando el modelo TUP [10] para evaluar software educacional. Las preguntas del cuestionario se dividen en cinco grupos: utilidad pedagógica, adecuación al contexto, facilidad de aprendizaje, interacción y utilidad en general. La escala usada para las respuestas de los estudiantes fue una escala de cuatro puntos (Desacuerdo- Parcialmente en desacuerdo-

Parcialmente de acuerdo- De acuerdo). Además, se permitió a los alumnos dar una opinión personal escrita sobre ciertos aspectos que se pudieran mejorar en el videojuego.

4.5.3 Resultados

Los resultados han sido divididos en los resultados obtenidos de la encuesta de valoración y los obtenidos en las opiniones personales de los estudiantes.

4.5.3.1 Resultados de la encuesta

Los resultados, que se encuentran registrados en la Tabla 4.2 donde se muestra el porcentaje de estudiantes que seleccionaron "Parcialmente de acuerdo" o "De acuerdo" en sus respuestas, muestran que *Evaluators* ha cosechado popularidad y éxito dentro de este grupo de alumnos como una herramienta complementaria para las clases de Procesadores de Lenguajes. Muchos estudiantes consideran que es divertida y útil para entender los diferentes aspectos relacionados con las gramáticas de atributos. Es interesante apuntar que los alumnos encuentran *Evaluators* interesante y, por lo tanto, lo recomendarían a sus compañeros de clase.

También se han detectado algunos aspectos de la herramienta que deben ser mejorados. Claramente, la funcionalidad de los oráculos debe mejorarse. Finalmente, clarificar qué instancia de atributo corresponde con qué copia de atributo en el inventario podría reducir los errores de los alumnos.

4.5.3.2 Resultados de la valoración libre

En cuanto a los resultados obtenidos en la valoración libre, se encuentran algunas valoraciones negativas en cuanto a aspectos técnicos del videojuego y a algunos *bugs*. El más importante, y que posiblemente tenga un fuerte impacto sobre los alumnos, es que los pasillos de algunos laberintos presentados tenían una longitud muy elevada, debido a ciertos problemas en el algoritmo inicial de dibujado de estos.

Tabla 4.2. Resumen de los resultados obtenidos en el cuestionario

Preguntas	**% de acuerdo**
Utilidad pedagógica	
Me ayuda a entender el procesamiento de las gramáticas de atributos	73,7%
Me ayuda a entender el papel de los atributos semánticos	73,7%
Me ayuda a entender el papel de las ecuaciones semánticas	68,4%
Me ayuda a distinguir los diferentes tipos de atributos semánticos	73,7%
No me resulta difícil darme cuenta de los errores que cometo	36,8%
Adecuación al contexto	
Evaluators complementa convenientemente las clases	84,2%
Evaluators me motiva a estudiar el procesamiento de las gramáticas de atributos	52,6%
Facilidad de aprendizaje	
Puedo empezar a usar la herramienta con soltura sin un largo periodo de entrenamiento.	63,2%
Los laberintos son una metáfora adecuada para representar árboles sintácticos	73,7%
El uso de mesas para recoger la información de los atributos me resulta natural	52,6%
Interacción	
He utilizado la pantalla del enunciado muchas veces	68,4%
He usado los oráculos	10,5%
Es fácil identificar que instancia de atributo corresponde con que copia de atributo en el inventario	31,6%
Es siempre fácil saber el estado de solución del problema	73,7%
Utilidad en general	
Evaluators es fácil de usar	42,1%
He disfrutado utilizando *Evaluators*	73,7%
Recomendaría *Evaluators* a otros alumnos de Procesadores de Lenguajes	68,4%

Por otro lado, también se proponen cambios en la mecánica de juego de *Evaluators* como la posibilidad de visualizar un mini-mapa del laberinto, y un sistema de anotaciones en el inventario para que el usuario se deje pistas a sí mismo y permitir el movimiento del avatar en modo mapa.

4.5.4 Acciones emprendidas

En base a los resultados obtenidos, se identificaron y prioritizaron una serie de mejoras. De éstas, la relativa al dibujado de los laberintos se estimó como una de las más relevantes, y, a la vez, más fácilmente resolubles, y fue, de hecho, resuelta en la versión actual de *Evaluators*, tal y como se describe en el capítulo anterior. Por otra parte, actualmente se está trabajando en mejorar la utilidad de los oráculos, añadiendo dinamismo e interactividad a los mismos, y se está debatiendo el resto de consideraciones extraídas de ambos estudios (encuesta de satisfacción y sugerencias de los estudiantes).

4.6 A Modo de Conclusión

A lo largo de este capítulo se ha descrito una evaluación formativa preliminar del sistema educativo *Evaluators*, basada en tres experiencias diferentes:

- La primera experiencia ha involucrado a docentes, y se ha centrado, en mayor medida, en el análisis de la potencial utilidad de la herramienta como instrumento de enseñanza. Gracias a la colaboración de los docentes se pudieron detectar distintas carencias en el sistema, tanto en el tipo de videojuego generado, como en la herramienta de autoría, carencias que se resolvieron para obtener la versión del sistema descrita en esta memoria.
- La segunda experiencia persiguió contrastar la eficacia de *Evaluators* como herramienta educativa, en la cual se ha deducido que el método propuesto es igual de efectivo que el método tradicional a la hora de enseñar los conceptos asociados a las gramáticas de atributos. Este resultado, si bien no espectacular, sí se considera realista y no negativo.
- Por último, se han detallado los resultados obtenidos de un estudio de la satisfacción con el sistema educativo del que se extrae, como principal

conclusión, que los alumnos encuentran la herramienta útil y entretenida para aprender los conceptos derivados del formalismo de las gramáticas de atributos.

Para concluir, comentar que los resultados obtenidos en el estudio de eficacia sitúan el método propuesto al mismo nivel que el método tradicional de enseñanza y práctica de este formalismo. Además, dicho método cuenta con la ventaja de la motivación extra que supone para los alumnos emplearlo, y que queda reflejada en el estudio de la satisfacción realizado.

Capítulo 5. Conclusiones y Trabajo Futuro

5.1 Conclusiones

A lo largo de este proyecto de fin de máster se ha presentado *Evaluators*, un sistema educativo diseñado para mejorar la experiencia educativa de los alumnos a la hora de aprender el modelo de proceso subyacente al formalismo de las gramáticas de atributos. Este sistema se basa en un generador de videojuegos, una herramienta de autoría y una herramienta de análisis que permiten al docente crear diferentes ejercicios que el alumno tendrá que resolver jugando al videojuego generado a partir de dichos ejercicios. Además, y gracias a la herramienta de análisis mencionada, el docente será capaz de evaluar el progreso del alumno. Por tanto, *Evaluators* se perfila como un sistema educativo con un gran potencial para conseguir mejorar la experiencia educativa de los estudiantes de las materias de Procesadores de Lenguajes en las primeras fases del curso.

Este sistema ha sido desarrollado siguiendo el modelo de proceso explicado en el capítulo 3. La característica principal de dicho modelo de proceso es que aboga por la producción de videojuegos educativos que están basados en ejercicios. Otra característica de este modelo es que permite una mejora de las herramientas software derivadas de una forma muy natural y sencilla, por ejemplo, en base a los resultados de evaluaciones formativas, tanto del grado de satisfacción como de la eficacia y eficiencia educativa, en las que pueden intervenir los distintos actores que están involucrados en el modelo de proceso.

De esta forma, se ha realizado una evaluación de las distintas herramientas software que componen el sistema educativo *Evaluators*. En base a esta evaluación se han conseguido dos objetivos esenciales:

- Mejorar la calidad de las herramientas creadas de una forma personalizada y ajustada a las necesidades reales de los actores más directamente involucrados con su uso.
- Medir preliminarmente la calidad de dichas herramientas en diferentes ejes (uso por parte del docente, uso por parte del alumno, etc.).

De la evaluación realizada se concluye que la eficacia educativa no desmerece la demostrada por los métodos tradicionales de enseñanza empleados en los conceptos tratados por el sistema, con la ventaja motivacional que ofrecen los videojuegos educativos.

5.2 Trabajo futuro

Como posibles líneas de trabajo futuro para continuar la investigación presentada en esta memoria pueden citarse las siguientes:

- Realizar una evaluación más extensa y continuada del sistema propuesto. Para ello, deberá plantearse una experiencia que implique la colaboración de los alumnos durante todo el curso para permitir observar el efecto a largo plazo del sistema en todo el proceso educativo.
- Estudiar la viabilidad de las sugerencias adicionales recogidas en la evaluación de la satisfacción del alumnado con *Evaluator* (véase la sección 4.5) e implementar aquellas que puedan resultar interesantes y útiles.
- Con el fin de perfilar al sistema como un producto final, crear un sitio web donde poder promocionar el sistema y ofrecer soporte a docentes y alumnos, además de crear una vía de comunicación, como, por ejemplo, un foro, donde los distintos usuarios de la herramienta puedan compartir sus experiencias y consejos con el sistema.
- Se está trabajando actualmente en la integración de *Evaluators* con la herramienta PAG descrita en el Capítulo 2.
- Por último, se está trabajando también en la construcción de un modelo de repositorio para objetos de aprendizaje en el dominio de los procesadores de lenguaje, en el que se espera integrar *Evaluators* como herramienta de producción de objetos. En [55][58] aparecen ya publicados algunos resultados preliminares relativos a esta línea de trabajo.

Bibliografía

1. ACM/IEEE. Computer Science Curriculum 2008: An Interim Revision of CS 2001. 2008
2. Aho, Alfred V. Teaching the compilers course. ACM SIGCSE Bulletin, 40(4):6-8,2008.
3. Aho, Alfred V.; Lam, Monica S.; Sethi, Rabi.; Ullman, Jeffrey D. Compilers: principles, techniques and tools (second edition). Addison-Wesley. 2007.
4. Aiken, Alexander. Cool: A portable project for teaching compiler construction. ACM SIGPLAN Notices, 31(7): 19-24, 1996.
5. Allenstein, Brett; Yost, Andrew; Wagner, Paul; Morrison, Joline. 2008. A query simulation system to illustrate database query execution. In Proceedings of the 39th SIGCSE technical symposium on Computer science education (SIGCSE '08). ACM, New York, NY, USA, 493-497.
6. Amory, A.; Naicker, K.; Vincent, J.; Adams, C. The Use of Computer Games as an Educational Tool: Identification of Appropriate Game Types and Game Elements. British Journal of Educational Technology. Vol. 30(4). (1999) 311-321.
7. Appel, Andrew W. Modern Compiler Implementation in Java. Cambridge University Press (New York, Cambridge). ISBN 0-521-58388-8 (hardback) 2008.
8. Aycock, John. The art of compiler construction projects. ACM SIGPLAN Notices, 38:28-32, December 2003.
9. Baldwin, Doug. A compiler for teaching about compilers. In SIGCSE'03: Proceedings of the 34th SIGCSE technical symposium on Computer science education, pages 220-223, New York, NY, USA, 2003. ACM.
10. Bednarik, R.; Gerdt, P.; Miraftabi, R.; Tukiainen, M. Development of the TUP Model - Evaluating Educational Software. ICALT'04 Proceedings of the 4th IEEE Int. Conference on Advanced Learning Technologies: 699-701
11. Bloom, B.; Furst, E.; Hill, W.; Krathwohl, D.R. Taxonomy of Educational Objectives: Handbook I, The Cognitive Domain, Addison-Wesley, 1956.
12. BOE número 187 de 4/8/2009, sección III, página 66699. BOE-A-2009-12977.
13. BOE número 278 de 20/11/1990, páginas 34401 a 34402 (2 págs.) BOE-A-1990-27912.
14. Bovet, Jean; Parr, Terence. ANTLRWorks: an antlr grammar development environment. Software: Practice and Experience, 38(12): 1305-1332, 2008.

15. Czarnecki, K.; Eisenecker, U. Generative Programming: Methods, Techniques and Applications. Addison-Wesley. 1999.
16. Daniel Resler; Dean Deaver. VCOCO: A Visualisation Tool for Teaching Compilers. In ITiCSE'98: Proceedings of the 6th annual Conference on the Teaching of Computing and the 3rd annual Conference on Integratin Technology into Computer Science Education, pages 199-202, New York, NY, USA,1998. ACM.
17. Demaille, Akim. Making compiler construction projects relevant to core curriculums. In ITiCSE'05: Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education, pages 266-270, New York, NY, USA, 2005. ACM.
18. Desherm, Herbert L.; McFall, Ryan L; Uti, Ngozi. Animation of Java linked lists. In Proceedings of the 33rd SIGCSE technical symposium on Computer science education (SIGCSE '02). ACM, New York, NY, USA, 53-57.
19. Eagle, Michael; Barnes, Tiffany. 2008. Wu's castle: teaching arrays and loops in a game. In Proceedings of the 13th annual conference on Innovation and technology in computer science education (ITiCSE '08). ACM, New York, NY, USA, 245-249.
20. Eagle, Michael; Barnes, Tiffany. 2009. Experimental evaluation of an educational game for improved learning in introductory computing. In Proceedings of the 40th ACM technical symposium on Computer science education (SIGCSE '09). ACM, New York, NY, USA, 321-325.
21. Elsworth, E.F.. The msl compiler writing project. ACM SIGCSE Bulletin, 24(2):41-44, 1992.
22. Fernández López, Rafael; Rodríguez Cerezo, Daniel; Valero Picazo, Ángel (2010) Knuthians: videojuego educativo para la enseñanza y el aprendizaje de las gramáticas de atributos. Proyecto de fin de carrera.
23. Fowler, Martin. Domain-Specific Languages. Addison-Wesley. 2011.
24. Garcia-Osorio, Cesar; Gomez-Palacios, Carlos; Garcia-Pedrajas, Nicolas. A Tool for Teaching LL and LR Parsing Algorithms. In ITiCSE'08: Proceedings of the 13th Annual Conference on Innovation and Technology in Computer Science education, pages 317-317, New York, NY, USA, 2008. ACM.

25. Garris, R.; Ahlers, R.; Driskell, J.E. Games, Motivation and Learning: A Research and Practice Model. Simulation & Gaming. Vol. 33(4). (2002) 441-467.
26. Gee, J.P. What video games have to teach us about learning and literacy. New York; Basingstoke: Palgrave Macmillan.(2003) 225 p.
27. Gómez-Martín, Marco Antonio. Arquitectura y metodología para el desarrollo de sistemas educativos basados en videojuegos. Tesis doctoral. 2007.
28. Gómez-Martín, Pedro-Pablo, Gómez-Martín, Marco Antonio; Palmier Campos, Pablo; González-Calero, Pedro A. Using Metaphors in Game-Based Education. Edutainment 2007: 477-488
29. GRAPPA. A Java Graph Package. http://www2.research.att.com/~john/Grappa/
30. Grinder, Michael T. 2002. Animating automata: a cross-platform program for teaching finite automata. In Proceedings of the 33rd SIGCSE technical symposium on Computer science education (SIGCSE '02). ACM, New York, NY, USA, 63-67.
31. Grinder, Michael T. 2003. A preliminary empirical evaluation of the effectiveness of a finite state automaton animator. In Proceedings of the 34th SIGCSE technical symposium on Computer science education (SIGCSE '03). ACM, New York, NY, USA, 157-161.
32. Griswold, Willian G. Teaching software engineering in a compiler project course. ACM Journal of Educational Resources in Computing, 2(4):3, 2002.
33. Hudson, Scott. CUP: LALR Parser generator for Java. http://www.cs.princeton.edu/~appel/modern/java/CUP/manual.html.
34. Jiménez-Díaz, G.; Gómez-Albarrán, M.; González-Calero, P-A. Pass the Ball: Game-Based Learning of Software Design. *ICEC'07 Proceedings of the Int. Conference on Entertainment Computing*: 49-54. 2007.
35. Jodar-Reyes, J-F; Revelles-Moreno, J. SEFALAS: Software para la Enseñanza de las Fases de Análisis Léxico y Análisis Sintáctico. http://lsi.ugr.es/plweb/static/software.html
36. Kaplan, Alan; Shoup, Denise. CUPV. A Visualization Tool for Generated Parsers. In SIGCSE'00: Proceedings of the thirty-first SIGCSE Technical Symposium on Computer Science Education, pages 11-15, New York, NY, USA, 2000. ACM.
37. Kleppe, Anneke. Software Language Engineering: Creating Domain-Specific Languages Using Metamodels. Addison-Wesley. 2008.

38. Knuth, Donald E. Semantics of Context-free Languages, Math.System Theory 2(2), 127-145, 1968.
39. Knuth, Donald E.. Semantics of Context-free languages: Correction. Math. Systems Theory, 5(1): 95-96, 1971.
40. Larraza-Mendiluze, Edurne; Garay-Vitoria, Nestor. 2010. Changing the learning process of the input/output topic using a game in a portable console. In Proceedings of the fifteenth annual conference on Innovation and technology in computer science education (ITiCSE '10).
41. Ledgard, Henry F. Ten mini-languages: A study of topical issues in programming languages. ACM Computing Surveys, 3(3): 115-146, 1971.
42. Leitner, S. 2003. So lernt man lernen. Der Weg zum Erfolg. Herder, Freiburg.
43. Mallozzi, John S.. Thoughts on and tools for teaching compiler construction. Journal of Computing in Small Colleges, 21(2):177-184, 2005.
44. Matínez-Ortiz, Iván; Sierra, José-Luis; Fernández-Manjón, Baltasar; Fernández-Valmayor Alfredo. Language Engineering Techniques for the Development of E-Learning Applications. Journal of Network and Computer applications 32(5) 1092 -1105. 2009.
45. Mernik, M.; Zumer, V.. An educational tool for teaching compiler construction. In IEEE Transactions on Education, volume 46, pages 61-68, 2003.
46. Moreno-Ger, Pablo; Sierra, José Luis; Martínez-Ortiz, Iván; Fernández-Manjón, Baltasar. A Documental Approach to Adventure Game Development. Science of Computer Programming 67(1) 3-31. 2007.
47. Natvig, Lasse; Line, Steinar. 2004. Age of computers: game-based teaching of computer fundamentals. In Proceedings of the 9th annual SIGCSE conference on Innovation and technology in computer science education (ITiCSE '04). ACM, New York, NY, USA, 107-111.
48. Navarro, Emily; Hoek, André van der. 2009. Multi-site evaluation of SimSE. In Proceedings of the 40th ACM technical symposium on Computer science education (SIGCSE '09). ACM, New York, NY, USA, 326-330.
49. Paakki, J. Attribute Grammar Paradigms - A High-Level Methodology in Language Implementation, ACM Computer Surveys, vol. 27, no. 2, pp. 196-255, 1995.

50. Parr, Terence. The Definitive ANTLR Reference: Building Domain-Specific Languages. 2007.
51. Reingold, Edward M.; Tilford, John S. Tidier Drawings of Trees. IEEE Transactions on Software Engineering. Vol. SE-7. Nº 2. 1981.
52. Robbins, Steven. 2007. A Java execution simulator. In Proceedings of the 38th SIGCSE technical symposium on Computer science education (SIGCSE '07). ACM, New York, NY, USA, 536-540.
53. Robbins, Steven. 2006. A UNIX concurrent I/O simulator. In Proceedings of the 37th SIGCSE technical symposium on Computer science education (SIGCSE '06). ACM, New York, NY, USA, 303-307.
54. Robbins, Steven. An Address Translation Simulator. In Proceedings of the 36th SIGCSE technical symposium on Computer science education (SIGCSE '05). ACM, New York, NY, USA, 515-519.
55. Rodríguez-Cerezo, D., Gómez-Albarrán, M., Sierra, J-L. Supporting Self-Regulated Learning in Technical Domains with Repositories of Learning Objects and Recommender Systems. International Workshop on Self-Regulated Learning in Responsive Open Learning Environments, in conjunction with ICALT 2011, Athens, Georgia, USA, 6-8 July 2011
56. Rodríguez-Cerezo, Daniel; Gómez-Albarrán, Mercedes; Sierra, José-Luis. From Collections of Exercises to Educational Games: A Process Model and a Case Study. ICALT'11: The 11th IEEE International Conference on Advanced Learning Technologies. 2011.
57. Ruckert, Martin. Teaching compiler construction and language design: making the case for unusual compiler projects with postscript as the target language. In SIGCSE'07: Proceedings of the twenty-ninth SIGCSE Technical Symposium on Computer Science Education, pages 232-236, New York, NY, USA, 1998. ACM.
58. Sarasa, A.; Rodríguez-Cerezo, D., Sierra, José-Luis. Ingeniería de Sistemas de Aprendizaje Electrónico y Esfuerzos de Estandarización: Un Caso de Estudio Relativo a los Almacenes de Objetos Didácticos. Novática 210, pp. 10-12. 2011. Publicado simultáneamente versión inglesa en UPGRADE, Vol. 2011, Nº 2, pp. 5-8.

59. Sathi, Harbans L. A project-based course in compiler construction. In SIGCSE'86: Proceedings of the seventeenth SIGCSE technical symposium on Computer science education, pages 114-119, New York, NY, USA, 1986. ACM.
60. Schreiner, Alex T.; Friedman, H. George. Introduction to Compiler Construction With Unix. Prentice-Hall. 1985.
61. Shaffer, David W.; Squire, Kurt R.; Richard, Halverson; Gee, James P. Video games and the future of learning. University of Wisconsin-Madison and Academic Advanced Distributed Learning Co-Laboratory. 2005.
62. Shapiro, Henry D.; Mickunas, M. Dennis. A new approach to teaching a first course in compiler construction. In SIGCSE' 76: Proceedings of the ACM SIGCSE- SIGCUE technical symposium on Computer science and education, pages 158-166, New York, NY, USA, 1976. ACM.
63. Sierra, José-Luis; Fernández-Pampillón. Ana Mª; Fernández-Valmayor, Alfredo. 2008. An environment for supporting active learning in courses on language processing. In Proceedings of the 13th annual conference on Innovation and technology in computer science education (ITiCSE '08). ACM, New York, NY, USA, 128-132.
64. Sierra, José-Luis; Fernández-Valmayor, Alfredo; Fernández-Manjón, Baltasar. A Document-Oriented Paradigm for the Construction of Content-Intensive Applications. Computer Journal 49(5) 562-584. 2006.
65. Sierra, José-Luis; Fernández Valmayor, Alfredo; Guinea, Mercedes; Hernanz, Héctor. From Research Resources to Learning Objects: Process Model and Virtualization Experiences. Journal of Educational Technology & Society 9(3) 56-68. 2006.
66. Sierra, José-Luis; Fernández-Manjón, Baltasar; Fernández-Valmayor, Alfredo. A Language-Driven Approach for the Design of Interactive Applications. Interacting with Computers, 20 (1) 112-127. 2008 .
67. Sierra, José-Luis; Fernández-Valmayor, Alfredo; Fernández-Manjón, Baltasar. From Documents to Applications using Markup Languages. IEEE Software 25(2) 68-76. 2008.
68. Stahl, T.; Voelter, M.; Czarnecki, K. Model-Driven Software Development: Technology, Engineering, Management, Wiley. 2006.
69. Sterling, Leon; Shapiro, Ehud. The Art of Prolog, 2nd Edition. 1994.

70. Urquiza-Fuentes, Jaime; Gallego-Carrillo, Micael; Gortázar-Bellas, Francisco; Velázquez-Iturbide, J. Ángel. Visualizing the symbol table. In ITiCSE'06: Proceedings of the 11th annual SIGCSE conference on Innovation and technology in computer science education, pages 341-341, New York, NY, USA, 2006. ACM.

71. Vegdahl, Steven R. Using visualization tools to teach compiler design. In Proceedings of the second annual CCSC on Computing in Small Colleges Northwestern conference, pages 72-83, USA, 2000. Consortium for Computing Sciences in Colleges.

72. Waite, William M., Jarrahian, Assad; Jackson, Michele H; Diwan, Amer. Design and implementation of a modern compiler course. In ITiCSE'06: Proceedings of the 11th annual SIGCSE conference on Innovation and technology in computer science education, pages 18-22, New York, NY, USA, 2006. ACM.

73. Werner, Michael. A parser project in a programming languages course. Journal of Computing in Small Colleges, 18(5):184-192, 2003.

74. Wetherell, Charles; Shannon, Alfred. Tidy Drawings of Trees. IEEE Transactions on Software Engineering. Vol. SE-5. Nº5. 1979.

75. White, Elizabeth; Sen, Ranjan; Stewart, Nina. Hide and show: using real compiler code for teaching. In Proceedings of the 36th SIGCSE technical symposium on Computer science education, SIGCSE'05, pages 12-16, New York, NY, USA, 2005. ACM.

76. White, Timothy M.; Way, Thomas P. 2006. jFAST: a java finite automata simulator. In Proceedings of the 37th SIGCSE technical symposium on Computer science education (SIGCSE '06). ACM, New York, NY, USA, 384-388.

77. Woolf, Beverly P. Building Intelligent Interactive Tutors: Student-centered strategies for revolutionizing e-learning. Morgan-Kaufman. 2008.

78. Xu, Li; Martin, Fred G. Chirp on crickets: teaching compilers using an embedded robot controller. In SIGCSE' 06: Proceedings of the 37th SIGSE technical symposium on Computer science education, pages 82-86, New York, NY, USA, 2006. ACM.

Apéndice A. Ejemplo de los archivos generados por el metagenerador de supuestos

Para ilustrar este ejemplo emplearemos la gramática especificada en la Figura 3.13 que define un lenguaje de una calculadora que permite sumar y multiplicar números. Además, cuenta con una memoria de constantes previamente inicializada. A continuación se muestran los principales archivos generados por el metagenerador de supuestos para construir el generador de supuestos acorde con este lenguaje. Estos son: un archivo JFLEX donde se describe el léxico del lenguaje y un archivo CUP donde se describe la parte sintáctica del lenguaje y se implementa el mecanismo que construirá el árbol sintáctico decorado y calculará el valor de cada una de las instancias de los atributos asociados a los nodos de dicho árbol.

En primer lugar se muestra el archivo JFLEX que será generado de la siguiente forma:

```
package MemCalc;


import java_cup.runtime.*;
%%
%unicode
%cup
%line
%column
%public
%class Scanner
%{
	public Scanner(java.io.InputStream r, DefaultSymbolFactory sf){
		this(r);
		this.sf=sf;
	}
	private DefaultSymbolFactory sf;
	public int yyline() {return yyline;}
	public int yycolumn() {return yycolumn;}
%}
%eofval{
	return sf.newSymbol("EOF",sym.EOF);
%eofval}

letter = [a-z]|[A-Z]|\_
posdigit = [1-9]
digit  = {posdigit}|0
num    = {posdigit}{digit}*
dec   = 0 | {digit}*{posdigit}
float =  {num}\.{dec}
iden   = {letter}({letter}|{digit})*
lambda = \<lambda\>
string = \"[^\"]*\"
%%
[\t\r\n ] {}
/*Terminales definidos en la gramatica*/
{num} {return sf.newSymbol("num",sym.num,yyline,yycolumn,new Token(yytext(),yyline,yycolumn));}
\) {return sf.newSymbol("cp",sym.cp,yyline,yycolumn,new Token(yytext(),yyline,yycolumn));}
\( {return sf.newSymbol("op",sym.op,yyline,yycolumn,new Token(yytext(),yyline,yycolumn));}
\* {return sf.newSymbol("mul",sym.mul,yyline,yycolumn,new Token(yytext(),yyline,yycolumn));}
\+ {return sf.newSymbol("add",sym.add,yyline,yycolumn,new Token(yytext(),yyline,yycolumn));}
```

```
{letter}({letter}|{digit})* {return sf.newSymbol("iden",sym.iden,yyline,yycolumn,new
Token(yytext(),yyline,yycolumn));}
\$ {return sf.newSymbol("end",sym.end,yyline,yycolumn,new Token(yytext(),yyline,yycolumn));}
[^] {throw new Error("Unrecognized character:"+yytext());}
```

Por último el archivo CUP generado para el ejemplo será:

```
package MemCalc;

import java_cup.runtime.*;
import java.util.HashMap;
import java.util.Stack;
import java.util.ArrayList;
import java.util.Iterator;
import Modelo.*;
parser code {:
	public HashMap<Atributo,Stack<Object>> depTable = new HashMap<Atributo,Stack<Object>>();
	public Object semanticClass = new semanticExpr();
	public int id=0;

	public Atributo getAtributo(Nodo nodo, String attr){
		boolean found =  false;
		Atributo result = null;
		int i = 0;
		ArrayList<Atributo> attrs = nodo.getAtributos();
		while(!found && i<attrs.size()){
			found = attrs.get(i).getNombre().compareTo(attr)==0;
			i++;
		}
		if(found){
			result = attrs.get(i-1);
		}
		return result;
	}

	public void solveTabDep(){
		boolean allComputed = false;
		while(!allComputed){
			Iterator<Atributo> itAttr = depTable.keySet().iterator();
			allComputed = true;
			while(itAttr.hasNext()){
				Atributo attr = itAttr.next();
				Stack<Object> pila = depTable.get(attr);
				if(pila.size()>0){
					Object obj = pila.pop();
					if(obj instanceof Pair){
						Pair par = (Pair) obj;
						int numArg = (Integer) par.getRight();
						String nameFunc = (String) par.getLeft();
						Object res = solveFunc(nameFunc, numArg,
							pila);
						if(res == null){
							pila.push(obj);
							allComputed = false;
						}else{
							pila.push(res);
							attr.setValor(res.toString());
						}
					}else if(obj instanceof Atributo){
						if(!((depTable.get((Atributo) obj).peek() instanceof
Pair) ||
(depTable.get((Atributo) obj).peek() instanceof Atributo))){
							pila.push(depTable.get((Atributo)
											obj).peek());
							allComputed = false;
						}else{
							pila.push(obj);
							allComputed = false;
					}
```

```
                                        }else if(obj instanceof String){
                                                attr.setValor((String) obj);
                                                pila.push(obj);
                                        }else if(obj instanceof Integer){
                                                attr.setValor(obj.toString());
                                                pila.push(obj);
                                        }else if(obj instanceof Float){
                                                attr.setValor(obj.toString());
                                                pila.push(obj);
                                        }else if(obj instanceof Object){
                                                attr.setValor(obj.toString());
                                                pila.push(obj);
                                        }
                                }
                        }
                }
        }

        public Object solveFunc(String nameFunc, int numArgs, Stack<Object> pila){
                ArrayList param = new ArrayList();
                Stack pilaAux = new Stack();
                boolean todoBien = true;
                int i = 0;
                while(todoBien && i<numArgs){
                        Object obj = pila.pop();
                        Object res = null;
                        pilaAux.push(obj);
                        if(obj instanceof Atributo){
                                if(((Atributo) obj).getValor() == null){
                                        todoBien = false;
                                }else {
                                        param.add(depTable.get(((Atributo) obj)).peek());
                                }
                        }else if(obj instanceof Pair){
                                Pair par = (Pair) obj;
                                res = solveFunc((String) par.getLeft(),(Integer)
                                                                        par.getRight(),pila);
                                if(res != null){
                                        param.add(res);
                                        pilaAux.pop();
                                        pilaAux.push(res);
                                }else{
                                        todoBien = false;
                                }
                        }else{
                                param.add(obj);
                        }
                        i++;
                }
                if(todoBien){
                        return Helper.invoke(semanticClass, nameFunc, param);
                }else{
                        while(!pilaAux.empty()){
                                pila.push(pilaAux.pop());
                        }
                        return null;
                }
        }
:}

// Declaración de Terminales
terminal Token num, cp, op, mul, add, iden, end;

// Declaración de no terminales
non terminal NodoNT exp;
non terminal NodoNT fact;
non terminal NodoNT s;
non terminal NodoNT term;
```

```
//Reglas de la gramática

s ::= exp : _ref_exp0 end : _ref_end0{:
        RESULT = new NodoNT(""+parser.id, "s", "s -> exp end", null);
        parser.id++;   Atributo s_val = new Atributo("val", RESULT);
        s_val.setDependeDe(new ArrayList<Atributo>());
        parser.depTable.put(s_val, new Stack<Object>());
        ArrayList<Nodo> childs = new ArrayList<Nodo>();
        childs.add(_ref_exp0);
        _ref_exp0.setPadre(RESULT);
        RESULT.setHijos(childs);
        NodoT terminal1 = new NodoT(""+parser.id, "end", null);
        parser.id++;   childs.add(terminal1);
        terminal1.setPadre(RESULT);
        Atributo lex_end1 = new Atributo("lex", terminal1);
        lex_end1.setValor("\""+_ref_end0.getLex()+"\"");
        lex_end1.setDependeDe(new ArrayList<Atributo>());
        parser.depTable.put(lex_end1, new Stack<Object>());
        parser.depTable.get(lex_end1).push(_ref_end0.getLex());
        RESULT.setHijos(childs);
        ArrayList<Atributo> dep = null;
        Nodo nodo = null;
        Atributo attr = null;
        Stack<Object> pila = null;
        nodo = RESULT;
        attr = s_val;
        dep = s_val.getDependeDe();
        pila = parser.depTable.get(attr);
        pila.push(parser.getAtributo(_ref_exp0, "val"));
        if(!dep.contains(parser.getAtributo(_ref_exp0, "val")))
                                        dep.add(parser.getAtributo(_ref_exp0, "val"));
        nodo = _ref_exp0;
        attr = parser.getAtributo(nodo, "mem_inh");
        if(attr != null){
                dep = attr.getDependeDe();
        }
        pila = parser.depTable.get(attr);
        pila.push(new Pair("initTable", 0));
:};

fact ::= num : _ref_num0{:
        RESULT = new NodoNT(""+parser.id, "fact", "fact -> num", null);
        parser.id++;   Atributo fact_val = new Atributo("val", RESULT);
        fact_val.setDependeDe(new ArrayList<Atributo>());
        parser.depTable.put(fact_val, new Stack<Object>());
        Atributo fact_mem_inh = new Atributo("mem_inh", RESULT);
        fact_mem_inh.setDependeDe(new ArrayList<Atributo>());
        parser.depTable.put(fact_mem_inh, new Stack<Object>());
        ArrayList<Nodo> childs = new ArrayList<Nodo>();
        NodoT terminal0 = new NodoT(""+parser.id, "num", null);
        parser.id++;   childs.add(terminal0);
        terminal0.setPadre(RESULT);
        Atributo lex_num0 = new Atributo("lex", terminal0);
        lex_num0.setValor("\""+_ref_num0.getLex()+"\"");
        lex_num0.setDependeDe(new ArrayList<Atributo>());
        parser.depTable.put(lex_num0, new Stack<Object>());
        parser.depTable.get(lex_num0).push(_ref_num0.getLex());
        RESULT.setHijos(childs);
        ArrayList<Atributo> dep = null;
        Nodo nodo = null;
        Atributo attr = null;
        Stack<Object> pila = null;
        nodo = RESULT;
        attr = fact_val;
        dep = fact_val.getDependeDe();
        pila = parser.depTable.get(attr);
        pila.push(lex_num0);
        if(!dep.contains(lex_num0)) dep.add(lex_num0);
        pila.push(new Pair("str2Int", 1));
:};
```

```
fact ::= iden : _ref_iden0{:
        RESULT = new NodoNT(""+parser.id, "fact", "fact -> iden", null);
        parser.id++;   Atributo fact_val = new Atributo("val", RESULT);
        fact_val.setDependeDe(new ArrayList<Atributo>());
        parser.depTable.put(fact_val, new Stack<Object>());
        Atributo fact_mem_inh = new Atributo("mem_inh", RESULT);
        fact_mem_inh.setDependeDe(new ArrayList<Atributo>());
        parser.depTable.put(fact_mem_inh, new Stack<Object>());
        ArrayList<Nodo> childs = new ArrayList<Nodo>();
        NodoT terminal0 = new NodoT(""+parser.id, "iden", null);
        parser.id++;   childs.add(terminal0);
        terminal0.setPadre(RESULT);
        Atributo lex_iden0 = new Atributo("lex", terminal0);
        lex_iden0.setValor("\""+_ref_iden0.getLex()+"\"");
        lex_iden0.setDependeDe(new ArrayList<Atributo>());
        parser.depTable.put(lex_iden0, new Stack<Object>());
        parser.depTable.get(lex_iden0).push(_ref_iden0.getLex());
        RESULT.setHijos(childs);
        ArrayList<Atributo> dep = null;
        Nodo nodo = null;
        Atributo attr = null;
        Stack<Object> pila = null;
        nodo = RESULT;
        attr = fact_val;
        dep = fact_val.getDependeDe();
        pila = parser.depTable.get(attr);
        pila.push(lex_iden0);
        if(!dep.contains(lex_iden0)) dep.add(lex_iden0);
        pila.push(parser.getAtributo(RESULT, "mem_inh"));
        if(!dep.contains(parser.getAtributo(RESULT, "mem_inh")))
                                dep.add(parser.getAtributo(RESULT, "mem_inh"));
        pila.push(new Pair("getValor", 2));
:};

fact ::= op : _ref_op0 exp : _ref_exp0 cp : _ref_cp0{:
        RESULT = new NodoNT(""+parser.id, "fact", "fact -> op exp cp", null);
        parser.id++;   Atributo fact_val = new Atributo("val", RESULT);
        fact_val.setDependeDe(new ArrayList<Atributo>());
        parser.depTable.put(fact_val, new Stack<Object>());
        Atributo fact_mem_inh = new Atributo("mem_inh", RESULT);
        fact_mem_inh.setDependeDe(new ArrayList<Atributo>());
        parser.depTable.put(fact_mem_inh, new Stack<Object>());
        ArrayList<Nodo> childs = new ArrayList<Nodo>();
        NodoT terminal0 = new NodoT(""+parser.id, "op", null);
        parser.id++;   childs.add(terminal0);
        terminal0.setPadre(RESULT);
        Atributo lex_op0 = new Atributo("lex", terminal0);
        lex_op0.setValor("\""+_ref_op0.getLex()+"\"");
        lex_op0.setDependeDe(new ArrayList<Atributo>());
        parser.depTable.put(lex_op0, new Stack<Object>());
        parser.depTable.get(lex_op0).push(_ref_op0.getLex());
        RESULT.setHijos(childs);
        childs.add(_ref_exp0);
        _ref_exp0.setPadre(RESULT);
        RESULT.setHijos(childs);
        NodoT terminal2 = new NodoT(""+parser.id, "cp", null);
        parser.id++;   childs.add(terminal2);
        terminal2.setPadre(RESULT);
        Atributo lex_cp2 = new Atributo("lex", terminal2);
        lex_cp2.setValor("\""+_ref_cp0.getLex()+"\"");
        lex_cp2.setDependeDe(new ArrayList<Atributo>());
        parser.depTable.put(lex_cp2, new Stack<Object>());
        parser.depTable.get(lex_cp2).push(_ref_cp0.getLex());
        RESULT.setHijos(childs);
        ArrayList<Atributo> dep = null;
        Nodo nodo = null;
        Atributo attr = null;
        Stack<Object> pila = null;
        nodo = _ref_exp0;
```

```
        attr = parser.getAtributo(nodo, "mem_inh");
        if(attr != null){
                dep = attr.getDependeDe();
        }
        pila = parser.depTable.get(attr);
        pila.push(parser.getAtributo(RESULT, "mem_inh"));
        if(!dep.contains(parser.getAtributo(RESULT, "mem_inh")))
                                        dep.add(parser.getAtributo(RESULT, "mem_inh"));
        nodo = RESULT;
        attr = fact_val;
        dep = fact_val.getDependeDe();
        pila = parser.depTable.get(attr);
        pila.push(parser.getAtributo(_ref_exp0, "val"));
        if(!dep.contains(parser.getAtributo(_ref_exp0, "val")))
                                        dep.add(parser.getAtributo(_ref_exp0, "val"));
:};

term ::= term : _ref_term1 mul : _ref_mul0 fact : _ref_fact0{:
        RESULT = new NodoNT(""+parser.id, "term", "term -> term mul fact", null);
        parser.id++;   Atributo term_val = new Atributo("val", RESULT);
        term_val.setDependeDe(new ArrayList<Atributo>());
        parser.depTable.put(term_val, new Stack<Object>());
        Atributo term_mem_inh = new Atributo("mem_inh", RESULT);
        term_mem_inh.setDependeDe(new ArrayList<Atributo>());
        parser.depTable.put(term_mem_inh, new Stack<Object>());
        ArrayList<Nodo> childs = new ArrayList<Nodo>();
        childs.add(_ref_term1);
        _ref_term1.setPadre(RESULT);
        RESULT.setHijos(childs);
        NodoT terminal1 = new NodoT(""+parser.id, "mul", null);
        parser.id++;   childs.add(terminal1);
        terminal1.setPadre(RESULT);
        Atributo lex_mul1 = new Atributo("lex", terminal1);
        lex_mul1.setValor("\""+_ref_mul0.getLex()+"\"");
        lex_mul1.setDependeDe(new ArrayList<Atributo>());
        parser.depTable.put(lex_mul1, new Stack<Object>());
        parser.depTable.get(lex_mul1).push(_ref_mul0.getLex());
        RESULT.setHijos(childs);
        childs.add(_ref_fact0);
        _ref_fact0.setPadre(RESULT);
        RESULT.setHijos(childs);
        ArrayList<Atributo> dep = null;
        Nodo nodo = null;
        Atributo attr = null;
        Stack<Object> pila = null;
        nodo = _ref_term1;
        attr = parser.getAtributo(nodo, "mem_inh");
        if(attr != null){
                dep = attr.getDependeDe();
        }
        pila = parser.depTable.get(attr);
        pila.push(parser.getAtributo(RESULT, "mem_inh"));
        if(!dep.contains(parser.getAtributo(RESULT, "mem_inh")))
                                        dep.add(parser.getAtributo(RESULT, "mem_inh"));
        nodo = RESULT;
        attr = term_val;
        dep = term_val.getDependeDe();
        pila = parser.depTable.get(attr);
        pila.push(parser.getAtributo(_ref_fact0, "val"));
        if(!dep.contains(parser.getAtributo(_ref_fact0, "val")))
                                        dep.add(parser.getAtributo(_ref_fact0, "val"));
        pila.push(parser.getAtributo(_ref_term1, "val"));
        if(!dep.contains(parser.getAtributo(_ref_term1, "val")))
                                        dep.add(parser.getAtributo(_ref_term1, "val"));
        pila.push(new Pair("def_MUL", 2));
        nodo = _ref_fact0;
        attr = parser.getAtributo(nodo, "mem_inh");
        if(attr != null){
                dep = attr.getDependeDe();
        }
```

```
        pila = parser.depTable.get(attr);
        pila.push(parser.getAtributo(RESULT, "mem_inh"));
        if(!dep.contains(parser.getAtributo(RESULT, "mem_inh")))
                                        dep.add(parser.getAtributo(RESULT, "mem_inh"));
:};

term ::= fact : _ref_fact0{:
        RESULT = new NodoNT(""+parser.id, "term", "term -> fact", null);
        parser.id++;   Atributo term_val = new Atributo("val", RESULT);
        term_val.setDependeDe(new ArrayList<Atributo>());
        parser.depTable.put(term_val, new Stack<Object>());
        Atributo term_mem_inh = new Atributo("mem_inh", RESULT);
        term_mem_inh.setDependeDe(new ArrayList<Atributo>());
        parser.depTable.put(term_mem_inh, new Stack<Object>());
        ArrayList<Nodo> childs = new ArrayList<Nodo>();
        childs.add(_ref_fact0);
        _ref_fact0.setPadre(RESULT);
        RESULT.setHijos(childs);
        ArrayList<Atributo> dep = null;
        Nodo nodo = null;
        Atributo attr = null;
        Stack<Object> pila = null;
        nodo = RESULT;
        attr = term_val;
        dep = term_val.getDependeDe();
        pila = parser.depTable.get(attr);
        pila.push(parser.getAtributo(_ref_fact0, "val"));
        if(!dep.contains(parser.getAtributo(_ref_fact0, "val")))
                                        dep.add(parser.getAtributo(_ref_fact0, "val"));
        nodo = _ref_fact0;
        attr = parser.getAtributo(nodo, "mem_inh");
        if(attr != null){
                dep = attr.getDependeDe();
        }
        pila = parser.depTable.get(attr);
        pila.push(parser.getAtributo(RESULT, "mem_inh"));
        if(!dep.contains(parser.getAtributo(RESULT, "mem_inh")))
                                        dep.add(parser.getAtributo(RESULT, "mem_inh"));
:};

exp ::= exp : _ref_exp1 add : _ref_add0 term : _ref_term0{:
        RESULT = new NodoNT(""+parser.id, "exp", "exp -> exp add term", null);
        parser.id++;   Atributo exp_val = new Atributo("val", RESULT);
        exp_val.setDependeDe(new ArrayList<Atributo>());
        parser.depTable.put(exp_val, new Stack<Object>());
        Atributo exp_mem_inh = new Atributo("mem_inh", RESULT);
        exp_mem_inh.setDependeDe(new ArrayList<Atributo>());
        parser.depTable.put(exp_mem_inh, new Stack<Object>());
        ArrayList<Nodo> childs = new ArrayList<Nodo>();
        childs.add(_ref_exp1);
        _ref_exp1.setPadre(RESULT);
        RESULT.setHijos(childs);
        NodoT terminal1 = new NodoT(""+parser.id, "add", null);
        parser.id++;   childs.add(terminal1);
        terminal1.setPadre(RESULT);
        Atributo lex_add1 = new Atributo("lex", terminal1);
        lex_add1.setValor("\""+_ref_add0.getLex()+"\"");
        lex_add1.setDependeDe(new ArrayList<Atributo>());
        parser.depTable.put(lex_add1, new Stack<Object>());
        parser.depTable.get(lex_add1).push(_ref_add0.getLex());
        RESULT.setHijos(childs);
        childs.add(_ref_term0);
        _ref_term0.setPadre(RESULT);
        RESULT.setHijos(childs);
        ArrayList<Atributo> dep = null;
        Nodo nodo = null;
        Atributo attr = null;
        Stack<Object> pila = null;
        nodo = _ref_term0;
        attr = parser.getAtributo(nodo, "mem_inh");
```

```
        if(attr != null){
                dep = attr.getDependeDe();
        }
        pila = parser.depTable.get(attr);
        pila.push(parser.getAtributo(RESULT, "mem_inh"));
        if(!dep.contains(parser.getAtributo(RESULT, "mem_inh")))
                                        dep.add(parser.getAtributo(RESULT, "mem_inh"));
        nodo = RESULT;
        attr = exp_val;
        dep = exp_val.getDependeDe();
        pila = parser.depTable.get(attr);
        pila.push(parser.getAtributo(_ref_term0, "val"));
        if(!dep.contains(parser.getAtributo(_ref_term0, "val")))
                                        dep.add(parser.getAtributo(_ref_term0, "val"));
        pila.push(parser.getAtributo(_ref_exp1, "val"));
        if(!dep.contains(parser.getAtributo(_ref_exp1, "val")))
                                        dep.add(parser.getAtributo(_ref_exp1, "val"));
        pila.push(new Pair("def_ADD", 2));
        nodo = _ref_exp1;
        attr = parser.getAtributo(nodo, "mem_inh");
        if(attr != null){
                dep = attr.getDependeDe();
        }
        pila = parser.depTable.get(attr);
        pila.push(parser.getAtributo(RESULT, "mem_inh"));
        if(!dep.contains(parser.getAtributo(RESULT, "mem_inh")))
                                        dep.add(parser.getAtributo(RESULT, "mem_inh"));
:};

exp ::= term : _ref_term0{:
        RESULT = new NodoNT(""+parser.id, "exp", "exp -> term", null);
        parser.id++;   Atributo exp_val = new Atributo("val", RESULT);
        exp_val.setDependeDe(new ArrayList<Atributo>());
        parser.depTable.put(exp_val, new Stack<Object>());
        Atributo exp_mem_inh = new Atributo("mem_inh", RESULT);
        exp_mem_inh.setDependeDe(new ArrayList<Atributo>());
        parser.depTable.put(exp_mem_inh, new Stack<Object>());
        ArrayList<Nodo> childs = new ArrayList<Nodo>();
        childs.add(_ref_term0);
        _ref_term0.setPadre(RESULT);
        RESULT.setHijos(childs);
        ArrayList<Atributo> dep = null;
        Nodo nodo = null;
        Atributo attr = null;
        Stack<Object> pila = null;
        nodo = _ref_term0;
        attr = parser.getAtributo(nodo, "mem_inh");
        if(attr != null){
                dep = attr.getDependeDe();
        }
        pila = parser.depTable.get(attr);
        pila.push(parser.getAtributo(RESULT, "mem_inh"));
        if(!dep.contains(parser.getAtributo(RESULT, "mem_inh")))
                                        dep.add(parser.getAtributo(RESULT, "mem_inh"));
        nodo = RESULT;
        attr = exp_val;
        dep = exp_val.getDependeDe();
        pila = parser.depTable.get(attr);
        pila.push(parser.getAtributo(_ref_term0, "val"));
        if(!dep.contains(parser.getAtributo(_ref_term0, "val")))
                                        dep.add(parser.getAtributo(_ref_term0, "val"));
:};
```

Apéndice B. Pre-Test realizado a los alumnos para la evaluación de la eficacia educativa de *Evaluators*

1. Indica las calificaciones que has obtenido en las siguientes asignaturas:

	No cursada o no presentado	SS	AP	NT	SB	MH
Introducción a la programación (IP)						
Laboratorio de programación I (LP1)						
Lógica (L)						
Matemática discreta (MD)						
Teoría de autómatas y lenguajes formales (TALF)						
Estructuras de datos y de la información (EDI)						
Laboratorio de programación II (LP2)						
Metodología y tecnología de la programación (MTP)						
Laboratorio de Programación III (LP3)						

2. Responde a las siguientes preguntas en relación con la asignatura “Procesadores de lenguajes”:

	No	Sí
¿Eres repetidor?		
¿Has hecho ya la práctica en cursos pasados?		

Tanto si eres repetidor como si no lo eres, ¿cuál consideras que es tu conocimiento de los siguientes aspectos de la asignatura, en una escala de 0 (nulo) a 5 (muy alto)?

	0	1	2	3	4	5
Especificación de los aspectos léxicos						
Especificación de los aspectos sintácticos						
Gramáticas de atributos						
Tablas de símbolos						
Especificación de las restricciones contextuales						
Especificación de la traducción						
Implementación de analizadores léxicos						
Implementación de traductores descendentes predictivos recursivos						
Implementación de estructuras de control						
Implementación de tipos construidos						
Implementación de procedimientos						
Prototipado de procesadores de lenguajes en Prolog: gramáticas de cláusulas definidas						
Herramientas específicas de construcción de procesadores de lenguajes: generadores de analizadores y traductores						
Implementación de traductores descendentes dirigidos por tablas						
Implementación de traductores ascendentes						

3. En relación con los videojuegos:

	No	Sí
¿Te gustan los videojuegos?		
¿Juegas habitualmente?		

Apéndice C. Ejercicios propuestos para la experiencia realizada

1) Ejercicio de prueba
   El lenguaje: ternas de números. El primer número representa la *carga* de la casa. El segundo número representa los metros cuadrados del ala izquierda de la casa, y el tercero los metros cuadrados del ala derecha.
   El procesamiento consiste en encontrar el *peso* de la casa. El peso es la *suma* de los pesos de ambas alas. El peso de cada ala es el producto de sus metros cuadrados por la *carga* de la casa.

   Gramática:

   Casa ::= Carga Ala Ala
      Ala.cargah = Carga.carga
      Casa.peso = Ala(0).peso + Ala(1).peso
   Carga ::= **num**
      Carga.carga = aNumero(**num**.lex)
   Ala ::= **num**
      Ala.peso = Ala.cargah * aNumero(**num**.lex)

   Frase: 30 10 20

2) El lenguaje: listas numéricas (ejemplo: [], [5,6], [5, [6,7]], …)
   Procesamiento: Encontrar la suma de todos los números que aparecen en la lista

   Gramática:

   Lista ::= []
      Lista.suma = 0

   Lista ::= [Elems]
     Lista.suma = Elems.suma

   Elems ::= Elems , Elem
     Elems(0).suma = Elems(1).suma + Elem.suma

   Elems ::= Elem
     Elems.suma = Elem.suma

   Elem ::= **num**
    Elem.suma = a Numero(**num.**lex)

   Elem ::= Lista
      Elem.suma = Lista.suma

   Frase: [2,3,[4,[],[5]]]

3) El lenguaje:
   - Una *sección de declaraciones* (opcional) seguida de # seguida de una *sección de ítems*
   - La sección de declaraciones consta de una secuencia de declaraciones separadas por ,.
   - Cada *declaración* es un *identificador* seguido de =, seguido de su valor numérico
   - La *sección de ítems* consta de una secuencia de ítems separados por ,
   - Cada *ítem* puede ser un identificador, o bien un número.

El procesamiento: Encontrar la suma de los valores de los ítems. Para los ítems numéricos, el valor viene dado por el propio número. Para los ítems *identificador*, su valor es el último indicado en la sección de declaraciones, o bien 0 en caso de que el identificador no haya sido declarado.
Para llevar a cabo este procesamiento se dispone de las siguientes funciones semánticas:

- nuevoAlmacen(): Crea un almacén que asigna valores numéricos a constantes.
- añadeCte(*cte,valor,almacen*): Construye un nuevo almacén substituyendo en *almacen* el valor de *cte* por *valor*, en caso de que *cte* ya aparezca en *almacen*, o, en otro caso, añadiendo *cte* con *valor* como su valor.
- valorDe(*cte*,*almacen*): Obtiene el valor de *cte* que figura en *almacen*, o bien 0 en caso de que *cte* no esté incluida en *almacen*.

Gramática:

S ::= Ctes # Items
  Items.ah = Ctes.a
  S.val = Items.val
Ctes ::= λ
  Ctes.a = nuevoAlmacen()
Ctes ::= LCtes
  Ctes.a = LCtes.a

LCtes ::= LCtes , Cte
  LCtes(0).a = añadeCte(Cte.id, Cte.v, LCtes(1).a)

LCtes ::= Cte
  LCtes.a = añadeCte(Cte.id, Cte.v, nuevoAlmacen())

Cte ::= **id** = **num**
  Cte.id = **id**.lex
  Cte.v = aNumero(**num**.lex)

Items ::= Items , Item
   Items(1).ah = Items(0).ah
   Item.ah = Items(0).ah
   Items(0).val = Items(1).val + Item.val

Items ::= Item
   Item.ah = Items.ah
   Items.val = Item.val

Item ::= **num**
  Item.val = aNumero(**num**.lex)

Item ::= **id**
  Item.val = valorDe(**id**.lex,Item.ah)

Frase:
    x = 5,
    y = 6
   #
   x, 7, y

4) El lenguaje:
   - Una *sección de actividades* seguida de #, seguida de una *sección de declaraciones*
   - La sección de actividades consta de una o más actividades separadas por ,.
   - La sección de declaraciones también consta de una o más actividades separadas por ,
   - Cada *actividad* es un identificador.

   El procesamiento: Comprobar que las frases están *bien formadas*, en el sentido de que toda actividad en la sección de actividades debe aparecer en la sección de declaraciones.
   Para llevar a cabo este procesamiento se dispone de las siguientes funciones semánticas:
   - nuevoAlmacen (): Crea un almacén vacío de actividades.
   - añadeActividad(*id, almacen*): Construye un almacén de actividades añadiendo la actividad identificada por *id* al almacén *almacen*.
   - estaEn(*id*,*almacen*): *Cierto* si *id* aparece en *almacén*; *falso* en otro caso.

   La gramática:
   S ::= As # Ds
     As.ah = Ds.a
     S.ok = As.ok

   As ::= As , A
     As(0).ok = As(1).ok **and** estaEn(A.id,As(0).ah)
     As(1).ah = As(0).ah

   As ::= A
      As.ok = estaEn(A.id,As.ah)

   Ds ::= A, Ds
     Ds(0).a = añadeActividad(A.id,Ds(1).a)

   Ds ::= A
     Ds.a = añadeActividad(A.id,nuevoAlmacen())

   A ::= **id**
     A.id = **id**.lex

   Frase:
   x, y, y, z # y, z, x

5) El lenguaje: Dos secuencias no vacías de números, separadas por &
   El procesamiento
   to: Multiplicar la suma de todos los números de la primera secuencia que no aparecen en la segunda y la suma de todos los números de la segunda secuencia que no aparecen en la primera.
   Para llevar a cabo este procesamiento se dispone de las siguientes funciones semánticas:
   - cVacio (): Crea un conjunto vacío de números.
   - añade(*n, c*): Construye el conjunto que resulta de añadir *n* a *c*.
   - estaEn(*n*,*c*): *cierto* si *n* está en *c*. *falso* en otro caso.

   La gramática:
   S ::= Nums & Nums
     S.val = Nums(0).val * Nums(1).val
     Nums(1).ph = Nums(0).p
     Nums(0).ph = Nums(1).p

   Nums ::= Nums, **num**

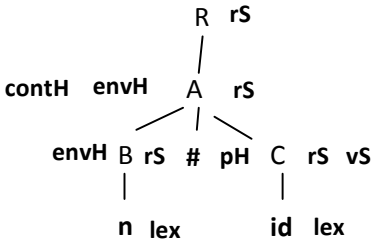
Nums(1).ph = Nums(0).ph
Nums(0).p = añade(**num**.lex, Nums(1).p)
Nums(0).val = **if**(estaEn(**num**.Nums(0).ph), 0, aNumero(**num**.lex))

Nums ::= **num**
Nums.val = **if**(estaEn(**num**.Nums.ph), 0, aNumero(**num**.lex))
Nums.p = añade(**num**.lex,cVacio())

Frase: 5,6,4,3,5 & 6,1,1,4,4

Apéndice D. Post-test realizado a los alumnos para evaluar la eficacia educativa de *Evaluators*

1) Considera el siguiente árbol de análisis sintáctico, en el que también se indican los atributos de cada nodo (los atributos heredados terminan en H, los sintetizados en S):



Determina cuáles de las siguientes afirmaciones son ciertas, y cuáles son falsas

Para calcular pH de C puede usarse envH de A	CIERTO ( )	FALSO ( )
Para calcular pH de C puede usarse envH de B	CIERTO ( )	FALSO ( )
Para calcular pH de C puede usarse rS de B	CIERTO ( )	FALSO ( )
Para calcular envH de B puede usarse lex de n	CIERTO ( )	FALSO ( )
Para calcular envH de A puede usarse contH de A	CIERTO ( )	FALSO ( )
Para calcular contH de A puede usarse envH de B	CIERTO ( )	FALSO ( )
Para calcular envH de A puede usarse rS de A	CIERTO ( )	FALSO ( )
Para calcular pH de C puede usarse rS de A	CIERTO ( )	FALSO ( )
Para calcular rS de C puede usarse vS de C	CIERTO ( )	FALSO ( )
Para calcular rS de A puede usarse envH de A	CIERTO ( )	FALSO ( )
Para calcular rS de B puede usarse envH de A	CIERTO ( )	FALSO ( )
Para calcular rS de B puede usarse lex de n	CIERTO ( )	FALSO ( )
Para calcular rS de C puede usarse envH de B	CIERTO ( )	FALSO ( )
Para calcular rS de C puede usarse rS de B	CIERTO ( )	FALSO ( )
Para calcular rS de C puede usarse rS de A	CIERTO ( )	FALSO ( )
La gramática tiene una producción A ::= B # C	CIERTO ( )	FALSO ( )

2) Considera una gramática de atributos definida sobre la siguiente gramática incontextual:

```
S ::= E
E ::= F
F ::= n
F ::= id
```

Sabiendo que tanto E como F tienen asociados un atributo heredado tH y un atributo sintetizado vS, determina cuáles de las siguientes afirmaciones son ciertas, y cuáles falsas

`E ::= id` debe tener asociada una ecuación de la forma E.tH = …	CIERTO ( )	FALSO ( )
`S ::= E` debe tener asociada una ecuación de la forma E.tH = …	CIERTO ( )	FALSO ( )
`F ::= id` debe tener asociada una ecuación de la forma F.vS =…	CIERTO ( )	FALSO ( )
`E ::= F` debe tener asociada una ecuación de la forma F.vS = …	CIERTO ( )	FALSO ( )
El orden en el que aparecen las ecuaciones asociadas a S ::= E determina el orden en el que se calculan los valores de los atributos	C IERTO ( )	FALSO ( )